TURING 图灵程序设计丛书

MANNING



Hello World!

Computer Programming for Kids and Other Beginners

父与子的编程之旅



[美] Warren Sande
Carter Sande 著
苏金国 易郑超 译

与小卡特一起学Python

人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

Warren Sande（父）

毕业于加拿大里贾那大学电气工程系。一直在面向计算机初学者教授软件基础课程，编写过大量广受欢迎的技术文档。

Carter Sande（子）

Warren之子，高中生，热爱计算机技术，自幼就跟着父亲玩编程，喜欢骑自行车和编写复古的电子游戏。



图灵程序设计丛书



Computer Programming for Kids and Other Beginners

父与子的编程之旅

与小卡特一起学Python



[美] Warren Sande
Carter Sande 著
苏金国 易郑超 译

人民邮电出版社
北京

图书在版编目（CIP）数据

父与子的编程之旅 ： 与小卡特一起学Python /（美）
桑德（Sande,W.），（美）桑德（Sande,C.）著 ；苏金国，
易郑超译. -- 北京 ： 人民邮电出版社，2014.12
（图灵程序设计丛书）
ISBN 978-7-115-36717-4

Ⅰ. ①父… Ⅱ. ①桑… ②桑… ③苏… ④易… Ⅲ.
①软件工具－程序设计 Ⅳ. ①TP311.56

中国版本图书馆CIP数据核字(2014)第183327号

内 容 提 要

本书是一本家长与孩子共同学习编程的入门书。作者是一对父子，他们以Python语言为例，详尽细致地介绍了Python如何安装、字符串和操作符等程序设计的基本概念，介绍了条件语句、函数、模块等进阶内容，最后讲解了用Python实现游戏编程。书中的语言生动活泼，叙述简单明了。为了让学习者觉得编程有趣，本书编排了很多卡通人物及场景对话，让学习者在轻松愉快之中跨入计算机编程的大门。

本书适合中小学生以及一切编程初学者。

◆ 著　　　[美] Warren Sande　Carter Sande
　译　　　苏金国　易郑超
　责任编辑　岳新欣
　责任印制　焦志炜
◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
　邮编　100164　　电子邮件　315@ptpress.com.cn
　网址　http://www.ptpress.com.cn
　北京　　　　　　印刷
◆ 开本：700×1000　1/16
　印张：28.25
　字数：559千字　　　　　2014年12月第1版
　印数：1－4 000册　　　　2014年12月北京第1次印刷

著作权合同登记号　图字：01-2014-4927号

定价：69.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

对本书第 1 版的赞誉

“一本同时适合小朋友和大朋友的好书。”

——Gordon Colquhoun，Avalon Consulting Services 公司计算机顾问

“这是一本为成长中的人们编写的 Python 书。”

——John Grayson 博士，*Python and Tkinter Programming* 作者

“这本书读起来很有意思，学起来也很有意思！”

——André Roberge 博士，加拿大圣安娜大学校长

“作者写了一本很友好也很有教育意义的编程书，学习起来很有趣也很轻松。”

——Bryan Weingarten，软件架构师

“我隆重推荐这本书！”

——Horst Jens，Python 教师，*Programming While Playing* 作者

“Python 是一种非常棒的编程入门语言。非常高兴看到这本特别为孩子写作的 Python 图书。”

——Jeffrey Elkner，教育工作者

“如果要教给孩子一件事，那就是原则。如果要教给孩子两件事，那就是原则和计算机编程。要教后者，只要有这本书就够了。”

——Josh Cronemeyer，ThoughtWorks 高级软件顾问

“我很喜欢这本书中与卡特的互动……我的学生们会非常喜欢那个电子宠物程序！这让我回想起自己多年前拥有的 Tamagotchi 电子宠物。”

——Kari J. Stellpflug，美国明尼苏达州罗切斯特公立学校教育工作者

“计算机编程是一种培养孩子学习能力的有力工具……学习编程的孩子会把这种能力也运用到其他东西的学习中。”

——Nicholas Negroponte，“每个孩子一台笔记本”计划发起人

对本书第 2 版的赞誉

“这本书将编程变得和煎培根一样容易。”

——Elisabet Gordon，Eagle Harbor 中学 10 年级学生

“对任何人来说，这都是一本非常出色的 Python 入门书。它非常有趣！”

——Mason Jenkins，Myron B. Thompson Academy7 年级学生

“上到 88 岁，下到 8 岁，任何想学习编程的人都可以阅读这本书。它不仅以一种有趣的方式介绍了 Python 编程，而且其中的最佳实践还适用于其他编程语言的学习。”

——Ben Ooms，Sogeti 公司软件工程师

“如果你想学习编程，或者想教孩子编程，那么这本书就是你的不二选择。”

——Cuberick.com

“不论老幼，只要想学习编程这门必备而有趣的技能，这都是一本非常好的介绍性图书。”

——Sue Gee，www.i-programmer.info 网站

“Warren 和 Carter 由简入难，直到教会读者制作有趣的 2D 图形游戏和模拟器。Python 是我向编辑新手推荐的首选语言，而本书正是非常好的学习资源。第 1 版出版后，我就一直向学生们推荐它。”

——Dave Briccetti，Dave Briccetti Software LLC 公司软件开发者和教师

推荐序一

我们为什么要“与小卡特一起学 Python”？

当我拿到《父与子的编程之旅：与小卡特一起学 Python》这本书时，非常巧，我正在琢磨怎么教我 8 岁的儿子编程。

当时为了找教材，我都做好了回一趟老家的打算——在一个旧书柜里，藏着二十多年前我参加全市组织的“初中生学编程”的所有讲义。那是我整个学生生涯中印象最深的教材了，而学习编程的岁月，也是我学生生涯中最深刻的记忆和最受益的历程。

30 年前，有位老人摸了摸小李劲的头，说了一句“计算机要从娃娃抓起”，当时绝大多数中国人还不知道计算机为何物。4 年后，正上初一的我得益于此，被选拔参加了全市组织的周末编程培训。乍一接触，便一头扎了进去，如痴如醉，即使刮风下雨感冒发烧也一节课没丢，至今脑海中还时不时浮现当时用循环编的由“*”组成的几何图形的画面。学编程不但没影响我的学习，反而给我带来了强大的自信。在获得全市最好的编程考试成绩后，追求卓越成为我的习惯，我的主课成绩也一直遥遥领先，就这样我非常顺利地度过了求学生涯。

说到培养孩子编程，有些人认为不宜太早，拔苗助长；有些人认为没用，因为在这个浮躁的当下，程序员已经沦为 IT 民工；还有些人觉得家长功利，以为是受新闻里那些少年出名的低龄程序员和黑客的影响。

先说功利。我是在非常严格的家庭中成长起来的，一路出类拔萃的成绩背后，是求学时每天 4 点起床早自习的辛苦和勤奋。为了从 99 到 100 那一分的提升，我付出了太多不必要的辛劳。所以在孩子的发展规划上，我没有任何情结，只是希望孩子不要重复我自己的弯路，能把他的天赋和能力发挥出来，做个有用的人，并无期待他“成名成家”的初衷。

再说说为什么孩子还这么小，我就着手教他学编程。无他，只为培养专一求精的精神。

现在的孩子大多一直生活在一个宽松、丰富多彩的成长环境下，钢琴、围棋、

机器人、网球、滑雪这些孩子们爱学爱玩的项目，我儿子也多有触及。但一直对他观察琢磨的我，发现了一个很大的问题，这也是现在孩子的通病：什么都玩一下，兴趣广泛，但对哪个都不通。不通非常致命，因为不通孩子就无法建立自信，没有自信就容易躲避困难，然后又更不自信，如此恶性循环。没有自信作为基础，孩子很难充分发挥自己的能力，形成自己的特长。所以说从小能有一个特长，有一个专注的、足够突出的拿手项目，对孩子的发展非常重要。

现在的孩子还有一个特点，就是非常聪明、精力旺盛。这个时候如果不能给他一个有乐趣、有难度的事，他的发展节奏就会被破坏，天赋就会被浪费。对于数理感觉比较好的孩子，编程其实是个非常好的选项。这里有创造、逻辑、条理、推理、计算，可以充分锻炼孩子。

至于学编程会不会变成 IT 民工命，我想这其实是一个“伪命题”。现在少年 MBA、少年领导力、情商培训很火，不是说它们没作用，但管理能力和情商在成长过程中可以潜移默化地培养，而具备一项专业的技能才是立身之根本。商界固然有马云、乔布斯这样没有专业技能的领袖，但更多能开创全新格局的还是有技术背景的业界领袖居多，如马化腾、李彦宏、Google 创始人、Facebook 创始人等。

我第一次接触 Python，是在十多年前。地球人都知道，编程语言是程序员表达思想、解决问题、创造乐趣的载体。程序员早期的生涯，都是以学习和掌握数种甚至数十种编程语言开始的。Linux 刚进中国的时候，我和一些编程同好热衷于汉化和自制 Linux 发行版。当时最大的 Linux 版本 Redhat 是我们研究和学习的首要对象，尤其是它的系统安装程序，灵活而复杂。很快，从 Redhat 某个早期版本中，我们发现了 Python 被用来实现了系统安装程序。那是我第一次学习 Python。灵活、表达能力强、面向对象，这是我对它的概括。现在有很多酷而强大的新语言，但 Python 已然拥有了庞大的用户基础，也成为了 Google 工程师们使用的三大语言之一。当年的苹果电脑标配了 Basic、绿色的字库界面，没有什么亲和力。而如今的 Python，天然和图形编程结合，想必会更得孩子们的喜欢。

跟孩子一起学编程，也是一个父亲的幸福时光。当我带着儿子完成 `print "Hello World"` 和 `print "1+2"` 后，他抢过电脑，想了一下，然后用稚嫩的小手输入了 `print "2345*10"`。计算机居然算对了！看到他惊讶的表情，我想每一位父亲都会享受到这种“成长的乐趣”。

陈俊
腾讯效果广告平台部商务研发中心总监
2014 年 10 月 12 日

推荐序二

1984 年 2 月 16 日，邓小平同志指出“计算机的普及要从娃娃抓起”。根据我自己多年中小学计算机教育和信息学奥林匹克竞赛辅导的经验，计算机科学对孩子智力发展的推动作用是十分明显的。现在，计算机科学已经不再是一门单独的学科，它不仅融合了各个学科发展的前进方向，也是各个学科研究的持续动力。此外，它对孩子的程序设计教育、数理逻辑培养、探究精神和创造性思维的培养，都是大有裨益的。

当前，市面上的计算机书籍汗牛充栋。然而，对于孩子来说，其中的绝大部分书籍都过于艰深。对于父母来说，陌生的内容使得他们无力更好地引导孩子学习。到头来，往往投入了大量的资金和时间，却收获甚微。

怎样才能在轻松愉快的学习过程中，让孩子体会到程序设计的趣味和魅力呢？你手中的这本书，便是一种能够让整个家庭充分享受编程乐趣的载体。

在内容编排上，这本书真正做到了深入浅出，循序渐进。跟随它学习，你会发现，编程课并不像那些教科书一般枯燥无味。通过富有趣味性的学习过程，孩子不仅可以巩固和提高自己的编程能力，也能在不知不觉中提升创造性的思维能力，了解编程内容中蕴含的逻辑思维。更重要的是，这是一本可以让孩子和父母共同阅读的书。无论多大年龄，无论之前是否有过计算机学习的经历，你都可以和孩子共同参与进来。我们深信，从简短的对家庭成员的问候程序，到和孩子共同设计的小游戏，整个学习过程不仅可以促进你与孩子之间的交流，也能让你从中受益良多。

作为一名多年从事程序设计教学的老师，更作为一个孩子的父亲，我谨在此将这本书推荐给每一个充满童真和好奇的孩子以及他们的父母，希望你们能够通过这本书了解编程，体会到 Python 程序设计的无穷魅力。

曹文

信息学奥林匹克竞赛金牌教练，全国模范教师

2014 年 10 月 16 日

第 1 版译者序

首先，你可能想知道这本书讲些什么。这是一本编程书，它会告诉你什么是编程，什么是程序，程序有哪些方面，需要了解哪些概念……我不想在这里列出这些深奥的术语把你吓住，你在书中可以找到，而且会发现其实这些概念一点也不深奥！最重要的是，读完这本书，你能自己编程序，甚至可以编写游戏，这可能是最让你着迷的一点吧。

也许你觉得这没有什么特别之处，不过作为译者，我从来没有这么热切地盼望一本书尽早出版，更确切地讲，应该说我女儿从来没有对我翻译的书表示出如此高涨的热情。因为，这本书确实与众不同！

你相信吗？这本书的作者之一卡特与你们一样，也是一个小学生，同样对计算机世界充满了好奇。也许你会惊喜地发现，你脑海中的疑问与他在书中问到的居然如出一辙。这本书不像一个糟糕的演讲者只顾自己长篇大论地说教，自以为作为听众的你已经领会他的意思；实际上，你会感觉卡特就像是你自己，你可以按自己的思维方式轻松地掌握书中的内容，可以发现你真正想问的问题并顺利找到答案，还可以在清晰的指导下动手编程，让大家对你刮目相看。

还等什么呢？现在就拿起书来，让它带你进入看似神秘的编程世界吧！不过不要忘了，一定要自己动手试一试，如果只是纸上谈兵，只看不做，你就无法感受到程序成功运行那一刻的快乐和成就感。

希望多年以后你在计算机领域小有成就时能这样感叹：多亏我小时候看过一本《父与子的编程之旅》，是一个小孩子和他的爸爸写的，那本书太棒了，要不是这本书……

本书由苏金国主译，姚曜、荆涛、高强、刘鑫、范松峰分别对全书各章进行审阅，另外乔会东，刘亮、王小振、李璜、牛亚峰等参与了全书的修改整理。全体人员共同完成了本书的翻译工作。特别要感谢苏钰涵小同学，作为这本书译稿的第一位小读者，她提出了很多宝贵的建议，正踌躇满志地着手开发自己的游戏……

前　　言

前言是什么？前言就是一本书开头的那一部分，这部分没多大意思，可以把前言跳过去直接读后面具体的内容。你是不是这么想的？确实，如果你真想这么干，当然可以跳过这个前言（喂，你是不是现在就打算翻页了？），不过天晓得你会漏掉什么好东西……反正篇幅也不长，也许你应该看看再说，没准真会有意想不到的收获。

什么是编程

很简单，编程（programming）就是告诉计算机要做什么。计算机只是一些没有生命的机器，它们自己可不知道该做什么，一切都得你来告诉它，而且你还必须把细节都说清楚。



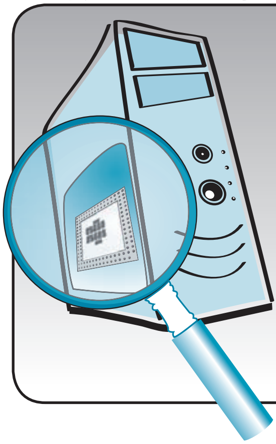
不过，一旦给计算机“下达”了正确的指令，它们就能做很多让人惊奇的事情。

术语箱

指令（instruction）就是下达给计算机的一个基本命令，通常要求计算机做某件特定的事情。

计算机程序是由多个指令组成的。为什么计算机能做到这么多了不起的事情呢？这是因为有许多聪明的程序员编写了程序或者软件（software）来告诉它们该怎样做。软件就是你的计算机上运行的程序，有时软件也可能运行在与你的计算机相连的另一台计算机上，比如 Web 服务器。

到底怎么回事？



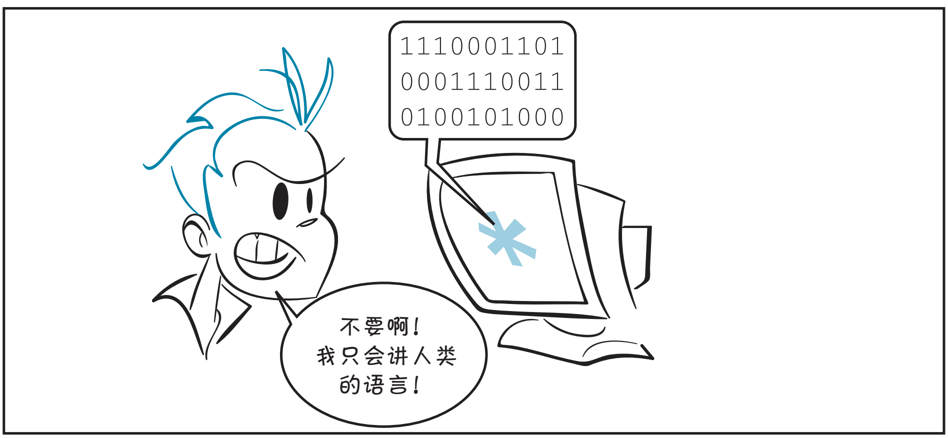
计算机要用非常非常多的电路来“思考”。在最底层，这些电路是一些开关。

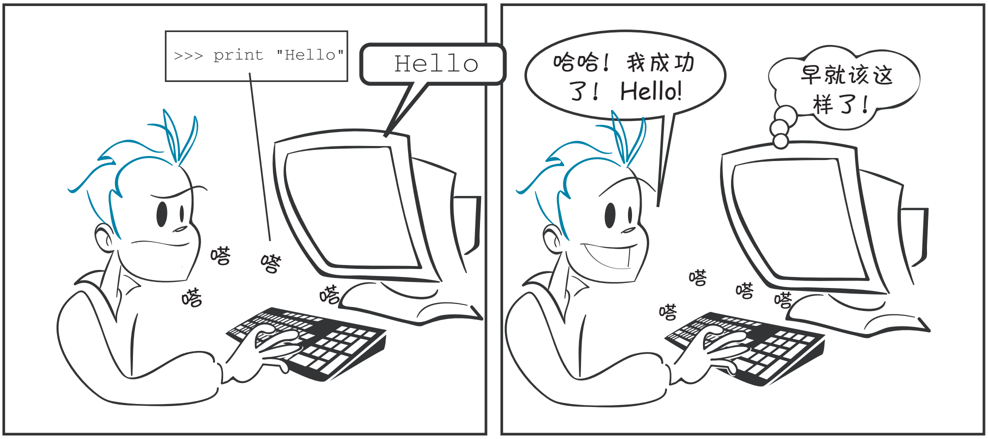
工程师和计算机科学家们使用 1 和 0 来代表“开”和“关”。所有这些 1 和 0 是一种二进制（binary）的编码。二进制实际上就表示“两种状态”。这两种状态分别是“开”和“关”，也就是 1 和 0。

你知道吗？ 二进制位 = 比特（bit）。

Python——我们和计算机沟通的语言

所有计算机在内部都使用二进制。不过大多数人都不擅长使用这种语言。我们需要一种更简便的方法来告诉计算机要做什么。所以人们发明了编程语言。利用计算机编程语言，我们可以先用一种自己能理解的方式写程序，然后再把它翻译成二进制供计算机使用。





有很多不同的编程语言。本书会教你如何使用其中的一种语言（Python）来告诉计算机要做什么。

为什么学编程

你可能不会成为一名专业的程序员（大多数人都不会），不过学习编程确实有很多理由。

- 最重要的原因是你想学！不论是作为业余爱好还是作为职业，编程都会很有意思，都会让你很有收获。
- 如果你对计算机感兴趣，想更多地了解它到底怎么工作，想知道怎样才能让它做你想做的事情，这也不失为学习编程的一个好理由。
- 也许你想编写自己的游戏，或者找不到合适的程序能完全满足你的需要，如果是这样，你就会想自己编写程序。
- 如今计算机已经无处不在，工作中、学校里或者在家里很有可能使用计算机（可能这三种场合都少不了计算机）。学习编程能帮助你从总体上更好地了解计算机。

为什么选用 Python 语言

既然有各种各样的编程语言可以选择（确实太多了！），对于这样一本给孩子们看的编程书，我为什么要选择 Python 呢？主要有以下几个原因。

- 最初创建 Python 语言的出发点就是为了便于学习。在我所见过的所有计算机语言中，Python 程序是最易读、最容易编写，也是最容易理解的。

- Python 是免费的。你可以下载 Python，还可以下载很多很多用 Python 编写的既好玩又有用的程序，所有这些都是免费的。
- Python 是开源（open source）软件。从某个角度来讲，“开源”的含义是指任何用户都可以扩展（extend）Python，也就是创建一些新“工具”。补充这些新工具后，就可以用 Python 做更多的事情，或者尽管是做同样的事情，但是有了这些新工具后会比原先更容易。很多人已经做了这种扩展，目前已经有非常多的免费 Python 工具可以供你下载。
- Python 并不是一个“玩具”。确实，它非常适合学习编程，不过实际上全世界每天都有成千上万的专业人士在使用这种语言，甚至包括类似 NASA（美国航空航天局）和 Google 这些机构的程序员。所以，学习 Python 后，你不用转换语言再去学一种“真正的”语言来编写“真正的”程序；很多工作都完全可以使用 Python 完成。
- Python 可以在各种不同类型的计算机上运行。Windows 电脑、苹果电脑和运行 Linux 的计算机上都可以使用 Python。大多数情况下，如果一个 Python 程序可以在你家里的 Windows 电脑上运行，那么这个程序同样也可以在你学校的苹果电脑上运行。本书适用于几乎所有安装了 Python 的计算机。（另外要记住，如果你要用的计算机上还没有安装 Python，完全可以免费安装。）
- 我自己很钟爱 Python，非常喜欢学习和使用这种语言，我想你也会和我一样。



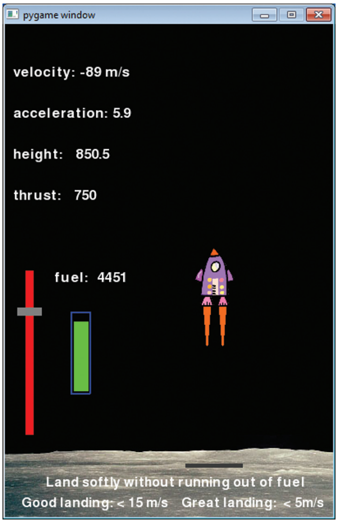
像程序员一样思考

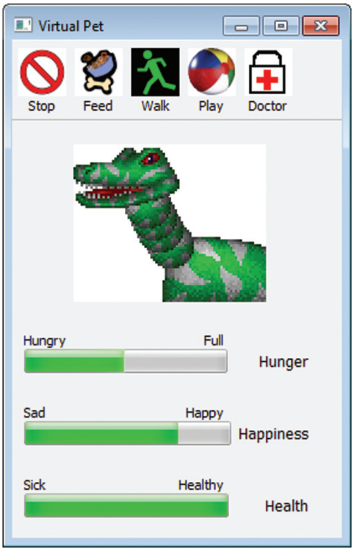
这本书会使用 Python 语言，不过你在这里学到的有关编程的大多数内容也适用于所有计算机语言。学习用 Python 编程可以让你有一个很好的起点，有了这个基础，将来学习任何其他语言都会很轻松。

有趣的内容

还有一点需要指出……

使用计算机最有趣的就是玩游戏，游戏中的图像和音效对小孩子尤其有吸引力。我们将要学习如何编写自己的游戏，在这个过程中还会利用图形和声音做很多工作。下面就是我们将要开发的一些程序的屏幕截图。







不过，我认为（或者说我希望），就像让飞船和滑雪的角色在屏幕上移动一样，你会发现学习这些基础知识并着手编写第一个程序同样很有趣。

祝你玩得开心！

关于本书

这本书讲的是计算机编程的基础知识。这是一本面向孩子们的书，不过只要想学习计算机编程，任何人都可以读这本书。

要看懂这本书，并不要求你之前对编程有任何了解，不过起码你要知道怎么使用计算机。也许你只是用计算机发邮件、上网、听音乐、玩游戏或者写学校布置的作业，但只要能在计算机上做一些基本的事情，比如说启动一个程序，打开和保存文件，学习这本书就绝对没问题。

你需要什么

本书会用一种名为 Python 的计算机语言教你学习编程。Python 是免费的，可以从很多地方下载，也包括本书的网站。要通过本书学习编程，你只需要具备如下条件。

- 这本书。（那当然了！）
- 一台计算机，已经安装了 Windows、Mac OS X 或者 Linux 操作系统。这本书中的例子都是在 Windows 上完成的。（对于 Mac 和 Linux 用户，还可以从这本书的网站 www.helloworldbook2.com 上得到一些帮助。）
- 使用计算机的一些基本知识（启动程序、保存文件等）。如果你在这方面有问题，可以找个人来帮你。
- 得到允许可以在你的计算机上安装 Python（可能是你的爸爸妈妈，也可能是你的老师，或者是负责这台计算机的某个人）。强烈建议你使用 Hello World 安装程序来安装使用本书所需的 Python 版本。该程序可在本书的网站 www.helloworldbook2.com 上找到。
- 渴望学习和尝试新事物，尽管需要多次尝试也不会轻易放弃的个性。

你不需要什么

通过本书学习编程，你不需要具备下列条件。

- 购买任何软件。你需要的一切都是免费的，而且本书的网站（www.helloworld-book2.com）上也提供了这些软件。
- 计算机编程的任何知识。这本书是面向初学者的。

怎样使用本书

如果想通过本书更好、更快地学习编程，要注意下面几点。

- 验证例子。
- 输入程序。
- 做习题。
- 别担心，放松点。

验证例子

下面就是本书例子的一个示例：

```
if timsAnswer == correctAnswer:
    print "You got it right!"
    score = score + 10
```

一定要按照例子自己重新做几遍并自己输入代码（我会明确地告诉你怎么做）。当然你也可以坐在一张舒适的大椅子上读完整本书，可能也能从中学到一些有关编程的知识。不过，通过自己动手编程，你学到的东西会多得多。

安装 Python

要想使用本书，你需要在自己使用的计算机上安装 Python。我们强烈建议你使用 Hello World 安装程序来安装本书所需的 Python 版本及其他内容。Hello World 安装程序可在本书的网站 www.helloworldbook2.com 上找到。

如果你采用其他方法安装 Python，导致没有安装上正确的 Python 版本以及所需的其他模块，那么当出现问题时，你会感到十分沮丧。

输入程序

本书提供的安装程序会把所有示例程序复制到你的硬盘上（如果你希望如此）。安装程序已经放在本书的网站上（www.helloworldbook2.com）。还可以从网站查看和下载单个例子，不过我建议你尽可能自己输入这些程序。通过亲手输入程序，你会对编程（特别是对 Python）产生一种“感觉”。（至少还可以多做一些打字练习！）

做习题

每一章的最后都有一些习题，可以练习你刚学到的知识。尽可能多地做些习题。如果你做不出来，可以找个懂编程的人来帮你。你们一起来解决这些问题，这样做会让你收获更多。做题之前千万别看答案，除非你实在做不出来了。（没错，有些答案在书的最后以及网站上已经给出，不过最好还是不要偷看。）

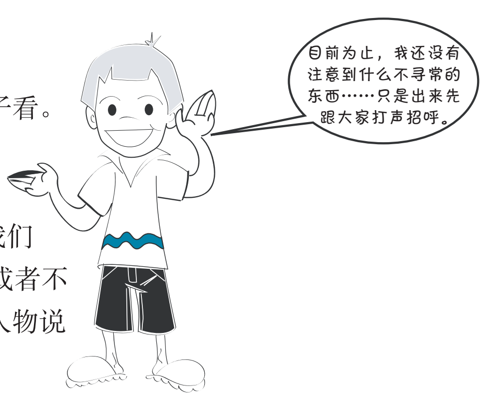
别担心，放松点！



不要担心犯错误。实际上，你得尽量多犯错误！我认为，犯错误然后搞清楚怎么找出错误并改正是最好的一种学习方法。

在编程中，除了多费一点时间，你的错误通常不会带来其他损失。所以完全可以犯很多错误，当然从中也会获得很多教训，你会发现这很有意思。

卡特说



我希望这本书有趣、易懂，适合小孩子看。很幸运，我有一个小帮手。卡特是一个小孩子，他热爱计算机，希望能更多地了解计算机。所以他能帮我保证这本书不偏离我们的初衷。卡特发现的有趣或不寻常的东西或者不合理的地方，在书中会通过右边这个卡通人物说出来。

第 2 版新增内容

首先，我们来说说第 2 版中有哪些内容没有改变，那就是我们决定在第 2 版中继续使用 Python 2，而不是转换到 Python 3。我们会在第 1 章解释其中的原因。

下面是第 2 版与第 1 版的不同之处：

- 增加了一些说明，解释 Python 2 和 Python 3 的区别。
- 第 12 章增加了一节关于 Python 字典的内容。
- 在第 20 章的 GUI 编程部分，我们将不再被支持的 PythonCard 换成了应用更广泛的 PyQt。PyQt 也用在第 22 章的 Hangman 程序和第 24 章的电子宠物程序中。
- 增加了第 25 章，用来详细解释第 10 章中的 Skier 程序。
- 增加了第 26 章，讲述在简单的对战游戏中，如何编写一个有人工智能（AI）的机器人来与其他机器人对战。
- 增加了一个附录，列出了 Python 2 和 Python 3 的不同之处。

致作者

你们可以在本书网站（www.helloworldbook2.com）上的作者在线论坛发表评论和提出问题。

致家长和老师

Python 是免费开源的软件，在计算机上安装和使用这种语言没有任何危险。Python 软件以及使用本书所需的所有软件都可以从 www.helloworldbook2.com 免费下载。这些下载文件很容易安装和使用，而且没有病毒和恶意插件。

第 1 版致谢

如果没有我的好妻子 Patricia，没有她给予的灵感、鼓励和支持，这本书根本不可能开始，当然也无从结束。因为卡特（我们的儿子）对学习编程产生了浓厚的兴趣，而我们找不到一本合适的书来满足他高涨的学习热情，所以 Patricia 对我说："你应该写本书，这会是一个不错的项目，你们两个可以合作来完成。"她总是对的，这一次也不例外。Patricia 总是有办法让人展示出最出色的一面。于是，卡特和我开始考虑这本书里该写些什么，我们一起构思每一章的大纲，编写示例程序，还想方设法力求更风趣、更有意思。一旦踏上征途，卡特和 Patricia 就坚信我们一定能胜利到达终点。卡特舍弃了每晚临睡前的故事时间，全心投入这本书。如果我们稍稍有一段时间放松，他就会提醒我："爸爸，我们好几天都没有写书了！"卡特和 Patricia 让我相信，只要你用心去做，没有做不到的事情。还要感谢家里的所有人，包括我们的女儿 Kyra，在我们写这本书时她也少了很多全家人在一起的欢聚时光。我要感谢家人的耐心和一如既往的支持，正是这一切才让这本书得以问世。

写稿是一回事，出版书又是另一回事。如果没有 Manning 出版公司 Michael Stephens 的热心和长久以来的支持，这本书绝不可能出版。从一开始，他就相当认可并赞同确实需要这样一种书。Michael 对这个项目充满信心，而且在整个过程中都一直耐心地指导我这样一个从来没有写过书的新手，这些对我们来说意义非比寻常，实在令人感激。我还要向 Manning 公司所有帮助我们完成这本书的人诚挚地道一声谢，特别是 Mary Piergies，感谢她耐心地协调制作过程的方方面面。

如果没有 Martin Murtonen 生动有趣的插图，这本书肯定会逊色不少。这些作品就能清楚地展示 Martin 过人的创造力和天赋。他还是一个非常容易相处的人，与他合作真是一件惬意的事情。

那一天，我问我的朋友（也是我的同事）Sean Cavanagh："要是用 Perl 来完成，你会怎么做？"Sean 回答说："我不会用 Perl，而是会用 Python。"于是我决定开始学习这种新的编程语言。在我学习 Python 期间，Sean 回答了我的很多问题，还仔细地审查了最初的书稿。他还创建并维护了这本书的安装程序。他的帮助让我感激不尽。

还要感谢在这本书出版过程中完成审校和帮助准备书稿的人们：Vibhu Chandreshekar、Pam Colquhoun、Gordon Colquhoun、Dr. Tim Couper、Josh Cronemeyer、Simon Cronemeyer、Kevin Driscoll、Jeffrey Elkner、Ted Felix、David Goodger、Lisa L. Goodyear、Dr. John Grayson、Michelle Hutton、Horst Jens、Andy Judkis、Caiden Kumar、Anthony Linfante、Shannon Madison、Kenneth McDonald、Evan Morris、Prof. Alexander Repenning、André Roberge、Kari J. Stellpflug、Kirby Urner 和 Bryan Weingarten，是他们的努力让这本书日臻完善。

Warren Sande

我要感谢 Martin Murtonen 专门给我画的漫画，感谢妈妈在我两岁的时候就让我玩计算机，而且还提出写书这样一个绝妙的想法。最重要的，我要感谢爸爸对这本书还有我付出的心血，感谢他教我学习编程。

Carter Sande

第 2 版致谢

在更新这本书的过程中，很多曾为第 1 版的问世作出贡献的人再次帮助了我。除了前面列出来的那些人以外，我们还要感谢帮忙审校本书第 2 版的人们：Ben Ooms、Brian T. Young、Cody Roseborough、Dave Briccetti、Elizabet Gordon、Iris Faraway、Mason Jenkins、Rick Gordon、Shawn Stebner 和 Zachary Young。此外还要感谢 Ignacio Beltran-Torres 和 Daniel Soltis，他们在这一版出版之前对最终的手稿做了非常仔细的技术校对。

最后，我们还要感谢 Manning 出版公司的所有员工，是他们让这一版比第 1 版更胜一筹。

目 录

第 1 章
出 发 吧

1.1 安装 Python

首先需要在你使用的计算机上安装 Python。

从前的美好时光



在个人计算机（PC）时代的初期，人们的日子很好过。最早的 PC 大都已经内置了一种名为 BASIC 的编程语言。人们什么也不必安装，他们要做的只是打开计算机，屏幕上会显示“READY”（准备就绪），然后就可以开始键入 BASIC 程序了。听上去很不错，是不是？

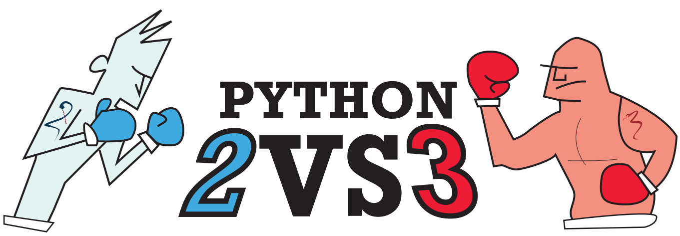
当然，那时能得到的也只有“READY”而已。没有程序，没有窗口，也没有菜单。如果你希望计算机做点其他的事情，就必须编写程序！那时没有字处理器、媒体播放器和 Web 浏览器，总之我们如今使用的所有应用当时都没有。甚至根本不存在万维网（Web），当然上网也就无从说起了。当时的计算机没有好玩的图片，也没有声音，只是在出错时偶尔会发出“哔哔”声！

安装 Python 非常容易。我们强烈建议你使用 Hello World 安装程序来安装使用本书所需的 Python 版本。该安装程序可在本书的网站 www.helloworldbook2.com 上找到。根据你的计算机的操作系统可以找到相应的安装程序版本。

这里分别提供了面向 Windows、Mac OS X 和 Linux 的版本。这本书里的所有例子都是 Windows 版本，不过在 Mac OS X 或 Linux 中使用 Python 也很类似。只需要按网站上的说明运行适合你的系统的版本。

即使你的计算机上已经安装了 Python，不打算使用这本书的安装程序，但是还要确保安装这本书需要的一些“额外内容”。查看网站（www.helloworldbook2.com）的安装说明（Installation Instructions）部分，看看应该怎么做。这里再强调一次，要想确保本书中的全部代码都能正确运行，最好的办法就是使用我们的安装程序。你可以在本书的网站（www.helloworldbook2.com）上找到该安装程序。

本书使用的 Python 版本是 2.7.3 版本。如果使用本书网站上的安装程序，你得到的就是这个版本。当你读到这本书时，可能已经有了更新的 Python 版本。这本书里的所有例子已经用 Python 2.7.3 做过测试。它们很可能也可以用于以后的 2.x 版本，不过我们无法预知未来，所以不能保证这一点。



Python 2 与 Python 3

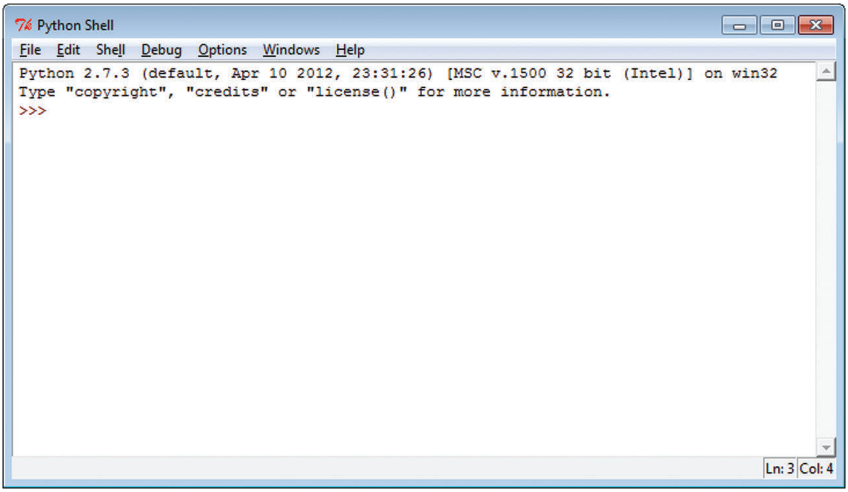
写作本书的前几年，Python 发布了一个新版本，也就是 Python 3。但是，它并不是 Python 一个真正意义上的“升级版本”。这就导致很多人并不想切换到 Python 3，所以他们仍然使用 Python 2。Python 的开发者也在同时开发 Python 2 的新版本和 Python 3 的新版本。在写作本书第 2 版的时候，Python 2 和 Python 3 的最新版本分别是 Python 2.7.3 和 Python 3.3.0。本书中使用的是 Python 2.7.3，这些代码应该能与 Python 2.x 的任何后续版本都兼容。

更多有关 Python 2 和 Python 3 的细节，请参阅附录 B。

1.2　从 IDLE 启动 Python

启动 Python 有两种方法。一种方法是从 IDLE 启动，也就是我们现在要使用的方法。

在 Start（开始）菜单中，可以看到“Python 2.7”下面的“IDLE (Python GUI)”。点击这个选项，会看到 IDLE 窗口打开（类似下面显示的窗口）。



IDLE 是一个 Python shell。shell 的意思就是“外壳”，基本说来，这是一个通过键入文本与程序交互的途径，可以利用这个 shell 与 Python 交互。（正是因为这个原因，可以看到窗口的标题栏上显示着 Python Shell）。IDLE 本身还是一个 GUI（图形用户界面），所以在开始菜单中显示为 Python GUI 。除了 shell，IDLE 还有其他一些特性，不过这个内容我们稍后再讲。

术语箱

GUI 就是图形用户界面（graphical user interface）。这表示界面中有窗口、菜单、按钮、滚动条等等。没有 GUI 的程序称为文本模式（text-mode）程序、控制台（console）程序或命令行（command-line）程序。

上图中的 >>> 是 Python 提示符（prompt）。提示符是程序等待你键入信息时显示的符号。这个 >>> 提示符就是在告诉你，Python 已经准备好了，在等着你键入 Python 指令。

1.3　来点指令吧

下面就来向 Python 下达我们的第一条指令。

在 >>> 提示符末尾的光标后面键入：

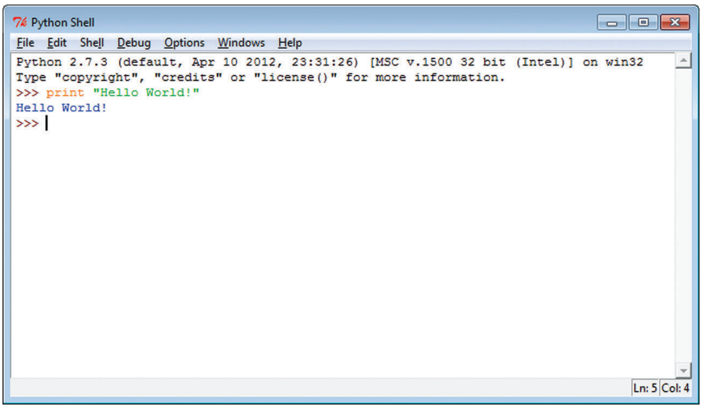
```
print "Hello World!"
```

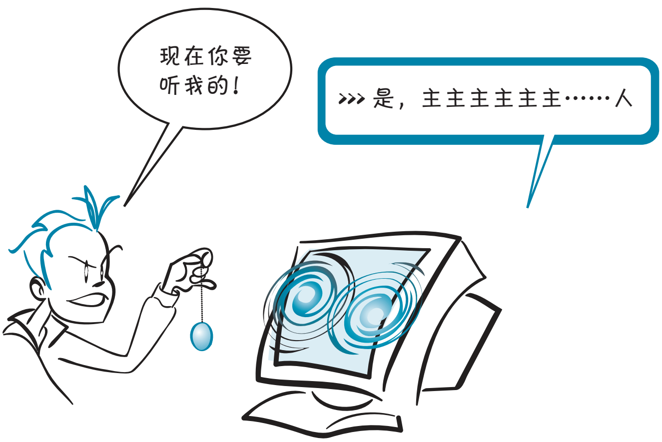
然后按下 Enter（回车键）。（有些键盘上，这个键称为 Return 键。）每键入一行指令之后，都要按回车键。

按下回车键之后，会得到这样一个响应：

```
Hello World!
>>>
```

下图显示了 IDLE 窗口中执行这个指令的情况。





Python 会完全照你说的去做：它会打印（print）你的消息。（在编程中，打印通常是指在屏幕上显示文本，而不是用打印机打印在一张纸上。）你键入的这行文本就是一个 Python 指令。你现在就是在编程！计算机已经在你的掌控之中！

另外，学习编程时总有这样一个传统：刚开始都是让计算机显示 “Hello

World!”。我们也会沿袭这个传统，这本书的书名就是从这里来的。欢迎来到编程世界！



这个问题问得好！IDLE 想帮你更好地理解这些内容。它用不同的颜色显示文本，便于你区分代码（code）的不同部分。（在 Python 之类的语言中，代码就是下达给计算机的指令，这只是指令的另一个叫法。）本书后面我会慢慢解释这些不同部分究竟是什么。

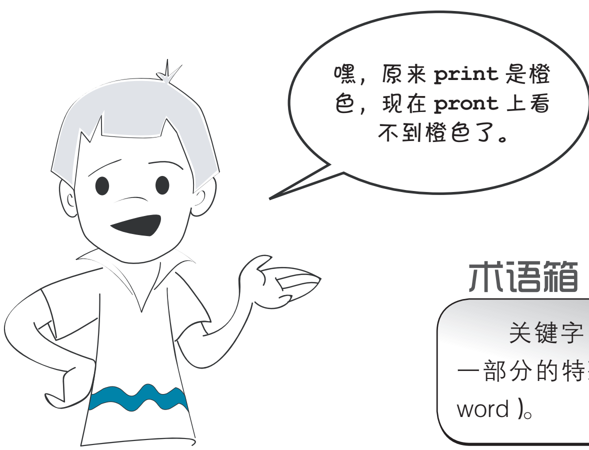
如果出问题

如果有错，可能会看到类似下面的结果：

```
>>> pront "Hello World!"
SyntaxError: invalid syntax
>>>
```

这个错误消息表示，Python 不懂你键入的内容。在上面的例子中，`print` 被错拼为 `pront`，Python 不知道该怎么处理。如果你犯了这个错误，可以再试一次，这一回一定要完全按照例子键入指令。





这是有道理的。因为 `print` 是一个 Python 关键字，而 `pront` 不是。

术语箱

关键字（keyword）是作为 Python 语言一部分的特殊词，也称为保留字（reserved word）。

1.4　与 Python 交互

你刚才所做的就是在交互模式中使用 Python。键入命令（指令）后，Python 立即执行这个命令。

术语箱

执行（executing）命令、指令或程序就表示“运行”或者“发生”，这只是运行或发生的另外一种形象说法。

下面就在交互模式中再尝试几条指令。

在提示符后面键入下面这条指令：

```
>>> print 5 + 3
```

你会得到：

```
8
>>>
```

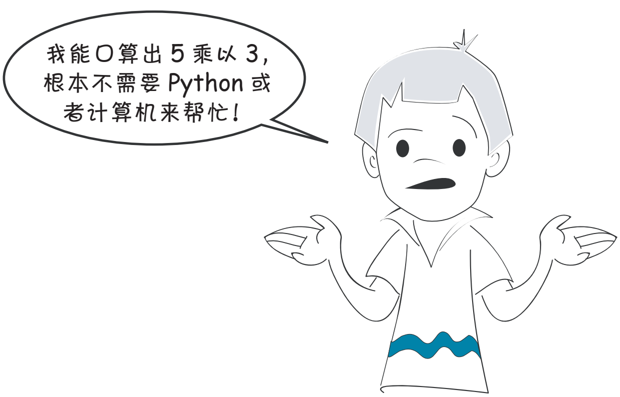
这么说 Python 确实会做加法！这并不奇怪，因为计算机本来就很擅长算术运算。

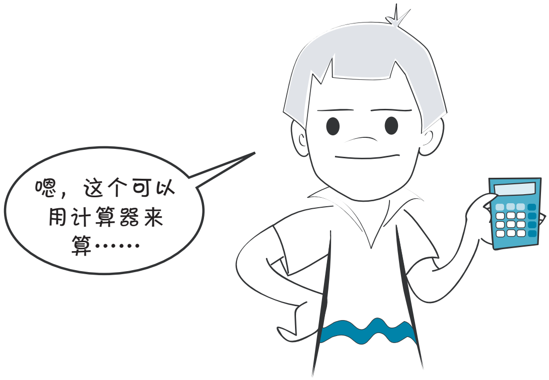
下面再试一个：

```
>>> print 5 * 3
15
>>>
```

几乎所有计算机程序和语言中都使用 * 符号作为乘号。这个符号称作“星号”或“星”。

如果你在数学课上总是把“5 乘以 3”写作 5 × 3，在 Python 中就必须习惯于用 * 来做乘法。（大多数键盘上，这个符号都在数字 8 的上面。）





那好，再试试这个：

```
>>> print 2345 * 6789
15920205
>>>
```

那么，这一个呢？

```
>>> print 1234567898765432123456789 * 9876543212345678987654321
12193263200731596000609652202408166072245112635269
>>>
```



没错。但是利用计算机，超大数的数学计算也能完成。不仅如此，你还可以做些别的事情，比如说：

```
>>> print "cat" + "dog"
catdog
>>>
```

或者再试试这个：

```
>>> print "Hello " * 20
Hello Hello Hello Hello Hello Hello Hello Hello Hello Hello
Hello Hello Hello Hello Hello Hello Hello Hello Hello Hello
```

除了数学计算，计算机擅长的另一件事就是反复地做事情。在这里，我们告诉 Python 让它把 Hello 打印 20 次。

后面还会在交互模式中做更多事情，不过现在……

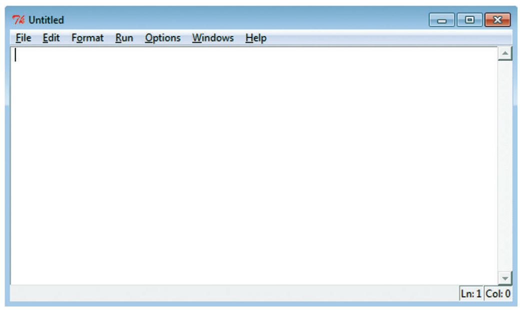
1.5 该编程了

到目前为止，我们看到的例子都只是（交互模式中）单个的 Python 指令。通过这些指令可以查看 Python 能够做些什么，这固然不错，不过这些例子并不是真正的

程序。前面已经提到过，程序是多个指令集合在一起。所以下面就来创建我们的第一个 Python 程序吧。

首先需要有办法键入我们的程序。如果只是在交互式窗口中键入指令，Python 不会“记住”你键入的内容。需要使用一个文本编辑器（比如 Windows 上的“记事本”、Mac OS X 上的 TextEdit，或者 Linux 上的 vi），它能把程序保存到硬盘上。IDLE 提供了一个文本编辑器，它比记事本更适合你的需要。可以从 IDLE 的菜单中选择 File（文件）➤ New Window（新窗口）找到这个文本编辑器。

> 谈到菜单选择时，比如说 File ➤ New，第一部分（这里的 File）是主菜单。由 ➤ 可以知道，下一部分（这里的 New）是 File 菜单中的一项。这本书中都将使用这种表示方法。



你会看到一个与右图类似的窗口。标题栏显示 Untitled（意思是“未命名”），因为你还没有给文件命名。

现在，在这个编辑器中键入代码清单 1-1 中的程序。

代码清单 1-1　我们第一个真正的程序

```
print "I love pizza!"
print "pizza " * 20
print "yum " * 40
print "I'm full."
```

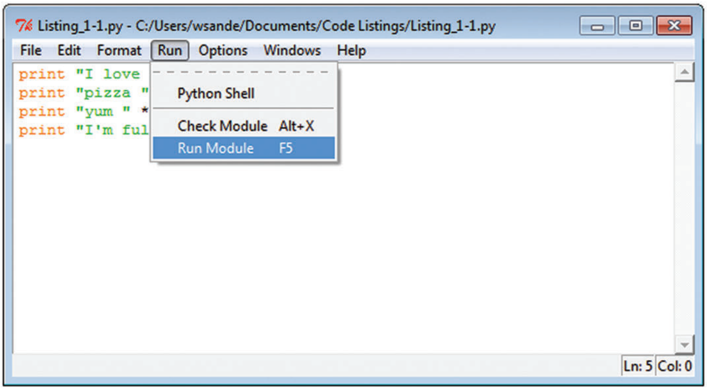
键入代码之后，使用 File（文件）➤ Save（保存）或者 File（文件）➤ Save As（另存为）菜单项保存这个程序。把这个文件命名为 pizza.py。你可以把它保存到你希望的任何位置（只要你记得保存在哪里，以便以后还能找到它）。你可能还想创建一个新的文件夹来保存你的 Python 程序。文件名末尾的 .py 部分很重要，因为这一部分会告诉你的计算机这是一个 Python 程序，而不只是普通的文本文件。

> 注意这个代码标题中出现了“代码清单 1-1”，这是什么意思？如果示例代码构成了一个完整的 Python 程序，我就会对它像这样编号，使你能很容易地在 \examples 文件夹或网站中找到相应的代码。

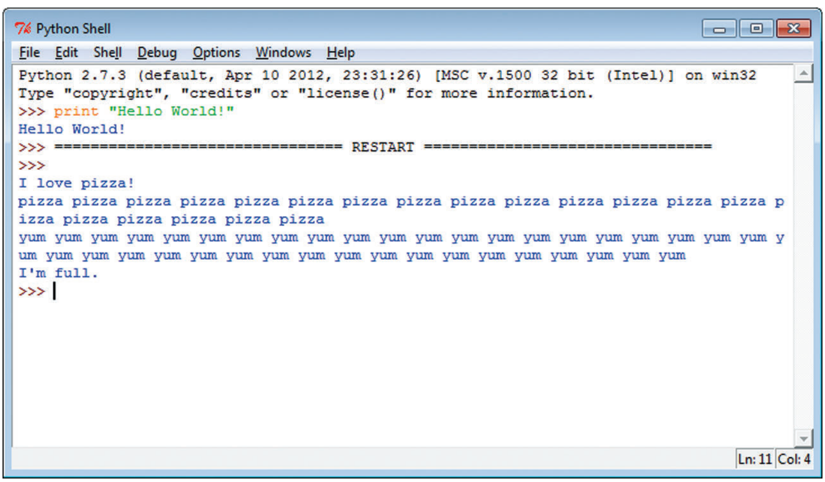
你可能已经注意到，这个编辑器在程序中使用了不同的颜色。有些词是橙色，还有一些是绿色。这是因为 IDLE 编辑器认为你打算键入一个 Python 程序。对于 Python 程序，IDLE 编辑器会把 Python 关键字用橙色显示，引号中间的所有内容都显示为绿色。这样是为了帮助你更容易地读 Python 代码。

1.6 运行你的第一个程序

保存了你的程序之后，就可以选择 Run（运行）菜单（还是在 IDLE 编辑器中），再选择 Run Module（运行模块），如下图所示。这样就能运行你的程序了。



Python Shell 窗口（就是启动 IDLE 时出现的那个窗口）再次变成活动窗口，并会显示下面的结果。



RESTART 部分表明已经开始运行一个程序。（如果你在反复运行程序来进行测试，这会很有帮助。）

然后程序开始运行。当然，这个程序确实没太大用处。不过起码你能让计算机听从你的号令了。随着学习的深入，我们的程序会越来越有意思。

1.7　如果出问题

如果程序中出现错误无法运行，怎么办呢？可能会发生两种不同类型的错误。下面就来了解这两种错误，这样无论遇到哪一种错误你都能知道如何应对。

语法错误

IDLE 在尝试运行程序前会对程序做一些检查。如果 IDLE 发现一个错误，这往往是一个语法错误（syntax error）。语法就是一种编程语言的拼写和文法规则，所以出现语法错误意味着你键入的某个内容不是正确的 Python 代码。

下面给出一个例子：

```
print "Hello, and welcome to Python!"
print "I hope you will enjoy learning to program."
print Bye for now!"
```

缺少引号

这里在 `print` 和 `Bye for now!"` 之间漏了一个引号。

如果运行这个程序，IDLE 会弹出一个消息“There’s an error in your program: invalid syntax.”，意思是说你的程序中有一个错误，语法不正确。你必须查看代码，找出哪里出了问题。IDLE 的编辑器会（用红色）突出显示它认为出错的位置。也许问题不会恰好出现在红色显示的位置，不过应该很接近。

运行时错误

可能发生的第二种错误是运行程序之前 Python（或 IDLE）无法检测出来的错误。这种错误只是在程序运行时才会发生，所以被称为运行时错误（runtime error）。下面是程序中出现运行时错误的例子：

```
print "Hello, and welcome to Python!"
print "I hope you will enjoy learning to program."
print "Bye for now!" + 5
```

如果保存这个程序，并试图运行，程序确实会开始运行。前两行会打印出来，但是接下来你会得到一个错误消息：

```
>>> ============================ RESTART ============================
>>>
Hello, and welcome to Python!
I hope you will enjoy learning to program.

Traceback (most recent call last):
  File "C:/HelloWorld/examples/error1.py", line 3, in <module>
    print "Bye for now!" + 5
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

错误消息开始

错误发生的位置

出错的代码行

Python 认为存在什么问题

Traceback 开头的代码行表示错误消息开始。下一行指出哪里发生了错误，这里会给出文件名和行号。然后显示出错的代码行，这可以帮助你找到代码中哪里出了问题。错误消息的最后一部分会告诉你 Python 认为存在什么问题。对编程和 Python 有了更多了解之后，就更容易理解这个消息是什么意思了。





听我说，卡特，这有点像将苹果和鳄鱼放在一起。在 Python 中，不能把两个完全不同的东西加在一起，比如说数字和文本。正是因为这个原因，print "Bye for now!" + 5 会给出错误消息。这就像是在说：“5 个苹果加 3 只鳄鱼是多少？”结果是 8，但是 8 个什么呢？把这些东西加在一起没有任何意义。不过几乎所有东西都可以乘以一个数来翻倍。（如果有两只鳄鱼，再乘以 5，那你就会有 10 只鳄鱼！）正因如此，print "Bye for now!" * 5 是可以的。

像程序员一样思考

看到错误消息也不用担心。它们只是为了帮助你找出哪里出了问题，以便你改正错误。如果程序中确实出了问题，你肯定更希望看到错误消息。没有给出任何错误消息的 bug[①] 才更难找到！



① bug，意思是“臭虫”。程序员通常把讨厌的错误说成 bug。——编者注

1.8　你的第二个程序

第一个程序没有多大实际意义，它只是在屏幕上打印了一些内容。下面来试一个更有意思的程序。

代码清单 1-2 中的代码编写的是一个简单的猜数游戏。与第一个程序一样，先选择 File（文件）➤ New Window（新窗口）在 IDLE 编辑器中新建一个文件。键入代码清单 1-2 中的代码，然后保存这个文件。可以把这个文件命名为你喜欢的任何名字，只要以“.py”结尾就可以。NumGuess.py 就是一个不错的名字。

这里只有 18 行 Python 指令，另外为了便于阅读还加入了一些空行。键入这些代码不会花费太多时间。虽然我们还没有说明这个代码到底是什么意思，不过不用担心，很快就会讲到。

代码清单 1-2　猜数游戏

```
import random
secret = random.randint(1, 99)          <—— 选一个秘密数
guess = 0
tries = 0
print "AHOY!  I'm the Dread Pirate Roberts, and I have a secret!"
print "It is a number from 1 to 99.  I'll give you 6 tries. "
while guess != secret and tries < 6:
    guess = input("What's yer guess? ")      <—— 得到玩家猜的数
    if guess < secret:
        print "Too low, ye scurvy dog!"
    elif guess > secret:
        print "Too high, landlubber!"

    tries = tries + 1                <—— 用掉一次机会
                                         （while 循环：最多允许猜 6 次）
if guess == secret:
    print "Avast! Ye got it!  Found my secret, ye did!"
else:
    print "No more guesses!  Better luck next time, matey!"
    print "The secret number was", secret
                                         （if/else 部分：游戏结束时打印消息）
```

键入这些代码时，注意 `while` 指令后面代码行是缩进的，另外 `if` 和 `elif` 后面的代码缩进得更多一些。还要注意有些代码行末尾有冒号。如果在正确的位置键入冒号，编辑器会自动将下一行缩进。

保存代码后，就像运行第一个程序一样，选择 Run（运行）➤ Run Module（运行模块）来运行这个程序。尝试一下，看看会发生什么。下面是我运行这个程序的示例：

```
>>> ======================= RESTART =======================
>>>
AHOY!  I'm the Dread Pirate Roberts, and I have a secret!
It is a number from 1 to 99.  I'll give you 6 tries.
What's yer guess? 40
Too high, landlubber!
What's yer guess? 20
Too high, landlubber!
What's yer guess? 10
Too low, ye scurvy dog!
What's yer guess? 11
Too low, ye scurvy dog!
What's yer guess? 12
Avast! Ye got it!  Found my secret, ye did!
>>>
```

我猜了 5 次才猜到这个秘密数，也就是 12。



后面几章我们会学习有关 while、if、else、elif 和 input 指令的所有内容。不过估计你已经大致了解了这个程序的基本过程了。

- 由程序随机选取秘密数。
- 用户输入他猜的数。
- 程序根据秘密数检查用户猜的结果：太大还是太小？
- 用户不断尝试，直到猜出这个数，或者用完所有机会。
- 猜到的数与秘密数一致时，玩家获胜。

你学到了什么

哇！内容真不少。这一章中，你做了下面这些事情：

- 安装了 Python；
- 学习了如何启动 IDLE；
- 了解了交互模式；
- 交给 Python 一些指令来执行；
- 看到了 Python 知道如何完成算术运算（包括非常大的数）；
- 启动 IDLE 文本编辑器键入你的第一个程序；

- 运行你的第一个 Python 程序；
- 了解错误消息；
- 运行你的第二个 Python 程序：猜数游戏。

测试题

1. 如何启动 IDLE ？
2. print 的作用是什么？
3. Python 中表示乘法的符号是什么？
4. 启动运行一个程序时 IDLE 会显示什么？
5. 运行程序又叫做什么？

动手试一试

1. 在交互模式中，使用 Python 计算一周有多少分钟。
2. 编写一个简短的小程序，打印 3 行：你的名字、出生日期，还有你最喜欢的颜色。打印结果应该类似这样：

```
My name is Warren Sande.
I was born January 1, 1970.
My favorite color is blue.
```

保存这个程序，然后运行。如果程序没有像你期望的那样运行，或者给出了错误消息，试着改正错误，让它能够正确运行。

第 2 章

记住内存和变量

什么是程序？嘿，等等，我想我们在第 1 章已经回答过这个问题！我们说过，程序就是下达给计算机的一系列指令。

对，确实是这样。不过，几乎所有真正有用或者有意思的程序都还有一些别的特征：

- ❒ 都有输入（input）；
- ❒ 都会处理（process）输入；
- ❒ 都会产生输出（output）。

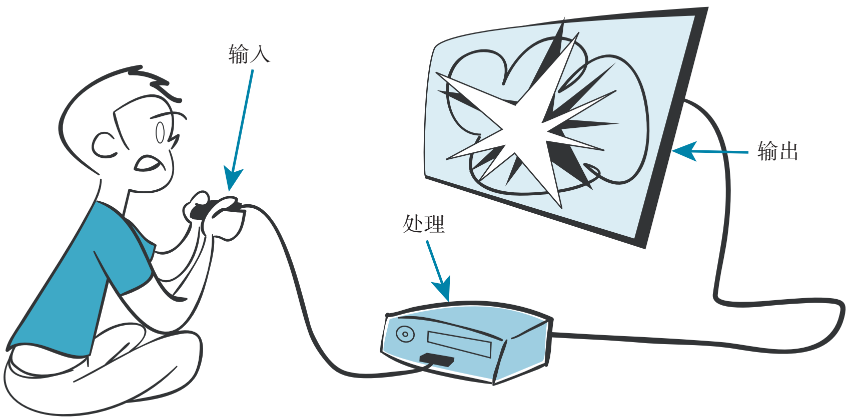
2.1　输入、处理和输出

你的第一个程序（代码清单 1-1）并没有任何输入或处理。也正是因为这个原因，那个程序没有太大意思。它的输出就是程序在屏幕上打印的消息。

你的第二个程序猜数游戏（代码清单 1-2）就具备以下这三个基本要素。

- ❒ 输入：玩家键入的数，也就是他猜的数。
- ❒ 处理：程序检查玩家猜的数，并统计已经猜过几次。
- ❒ 输出：程序最后打印的消息。

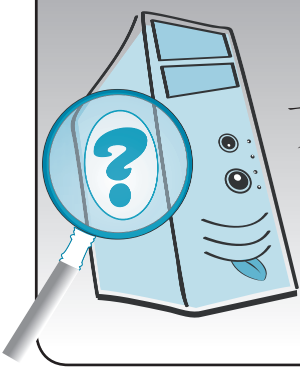
下面再看一个例子，这个程序也具备所有这三个基本要素：在一个视频游戏中，输入是来自操纵杆或游戏控制器的信号，处理是程序确定你是否击中外星人、避开火球、顺利过关或者做其他活动，输出是屏幕上显示的图形和扬声器或耳机传出的声音。



输入、处理和输出。一定要把这些记住。

那好，这么说计算机需要输入。不过它会怎么处理这些输入呢？为了处理输入，计算机必须记住它们，或者把它们保存在某个地方。计算机会把这些内容（包括输入以及程序本身）保存在它的内存中。

到底怎么回事？



你可能听说过计算机内存，不过这到底是什么意思呢？

我们说过，计算机只是一大堆开关。不错，内存就像是放在同一个位置上的一组开关。一旦以某种方式设置了这些开关，它们就会一直保持那种状态，直到你做出改变。也就是说，它们会记住你原先的设置……

哇，内存！

你可以写内存（设置开关），或者读内存（查看开关如何设置，不过不做任何改变）。

但是我们怎么告诉 Python 要把一个东西放在内存中的某个位置呢？另外，放在那里之后，又怎么再把它找回来呢？

在 Python 中，如果希望程序记住某个东西，以便你以后使用，所要做的就是给这个“东西”起一个名字。Python 会在计算机的内存中为这个“东西”留出位置，可能是数字、文本、图片或者音乐。下次想要引用这个东西时，只需要使用同一个名字。

下面还是在交互模式中使用 Python，对名字多做一些研究吧。

2.2 名字

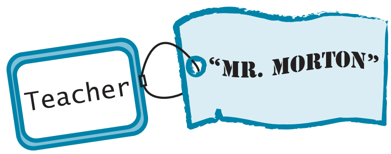
再回到 Python Shell 窗口。（如果完成第 1 章中的例子后关闭了 IDLE，现在要再打开它。）在提示符后面键入：

```
>>> Teacher = "Mr. Morton"
>>> print Teacher
```

（记住，>>> 是 Python 显示的提示符。你只需要键入它后面的内容，然后按回车。）你会看到下面的结果：

```
Mr. Morton
>>>
```

你刚才创建了一个由字母 "Mr. Morton" 组成的东西，并且给它起了一个名字 Teacher。



这里的等号（=）告诉 Python 要赋值（assign）或者“让……等于……”。这里把字母序列 "Mr.Morton" 赋值给名字 Teacher。



在计算机内存中的某个位置，字母序列 "Mr.Morton" 已经存在。你不需要准确地知道它们到底在哪里。只需要告诉 Python 这个字母序列的名字是 Teacher，从现在开始就要通过这个名字来引用这个字母序列。名字就像标签或者不干胶便条，你可以用它来标识一些东西。

在一个东西两边加上引号时，Python 会按字面来处理它。它会把引号里的内容原样打印出来。如果没有加引号，Python 就必须明确这个东西到底是什么。这可能是数字（如 5）、表达式（比如 5 + 3）或者名字（如 Teacher）。由于我们创建了名字 Teacher，所以 Python 会打印这个名字里的内容，这正是字母序列 "Mr. Morton"。

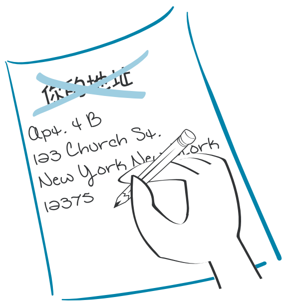
这就像有人在说，“请写下你的地址”，你肯定不会这样写（如右图）：





（不过，也许卡特会这么干，因为他总是喜欢调皮捣蛋……）

你可能会这样写：



如果写成“你的地址”，就是在按字面看这句话。除非加上引号，否则 Python 不会按字面来处理。下面来看另一个例子：

```
>>> print "53 + 28"
53 + 28
>>> print 53 + 28
81
```

有引号时，Python 会直接照你所说显示输出：53 + 28。

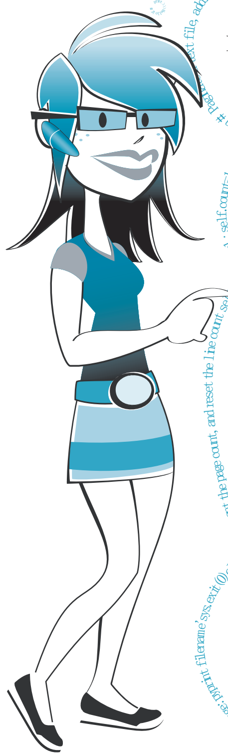
没有引号时，Python 把 53 + 28 处理为一个算术表达式，它会计算这个表达式。在这里，这是一个两数相加的表达式，所以 Python 会给出它们的和。

术语箱

算术表达式（arithmetic expression）是数字和符号的一个组合，Python 可以算出它的值。

计算（evaluate）就表示“算出……的值”。

Python 要确定需要多少内存来存储这些字母，以及要使用哪一部分内存。要获取信息（取回信息），只需要再使用同样的名字。我们使用 `print` 关键字并提供名字，这会在屏幕上显示具体的内容（如数字或文本）。



像程序员一样思考

把一个值赋给一个名字时（如把值“Mr. Morton”赋给 Teacher），它会存储在内存中，称为变量（variable）。在大多数编程语言中，都把这称为“把值存储（store）在变量中”。

不过 Python 与大多数其他计算机语言的做法稍有不同。它并不是把值存储在变量中，而更像是把名字放在值上。

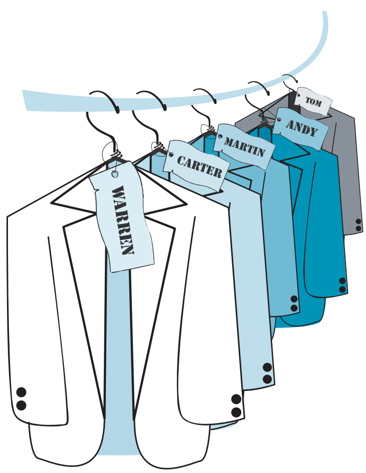
有些 Python 程序员说，Python 没有“变量”，而是只有“名字”。不过其实它们的行为基本上是一样的。这是一本关于编程的书（只不过刚好使用了 Python），而不是专门讨论 Python 的。所以谈到 Python 名时，我们可能会交替使用变量（variable）、名字（name）或变量名（variable name）等不同的术语。实际上，只要你理解变量有什么表现以及在程序中如何使用变量，怎么称呼并不重要。

顺便说一句，Guido van Rossum（也就是创建 Python 的那个人）在他的 Python 教程中曾经这样讲：“‘=’符号用来将一个值赋给一个变量。”所以我猜他认为 Python 是有变量的！

一种简洁的存储方法

在 Python 中使用名字就像是干洗店给衣服加标签。你的衣服挂在晾衣架上，上面附着写有你的名字的标签，这些衣服都挂在一个巨大的旋转吊架上。你回来取衣服时，并不需要知道它们存放在这个大型吊架的具体哪个位置。只需要提供你的名字，干洗店的人就会把衣服交还给你。实际上，你的衣服可能并不在原先所放的位置。不过干洗店的人会为你记录衣服的位置。要取回你的衣服，只需要提供你的名字。

变量也一样。你不需要准确地知道信息存储在内存中的哪个位置。只需要记住存储变量时所用的名字，再使用这个名字就可以了。



除了字母，还可以为其他内容创建变量。可以给数值指定名字。你应该还记得前面的例子：

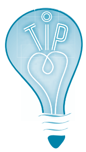
```
>>> 5 + 3
8
```

下面用变量来完成这个例子：

```
>>> First = 5
>>> Second = 3
>>> print First + Second
8
```

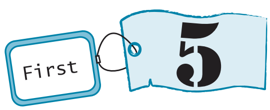
在这里，我们创建了两个名字 First 和 Second。数字 5 赋给 First，数字 3 赋给 Second。然后用 print 把这两个数的和打印出来。下面是完成这个例子的另一种做法。你可以试试看：

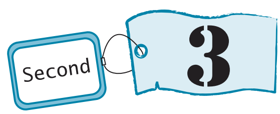
```
>>> Third = First + Second
>>> Third
8
```

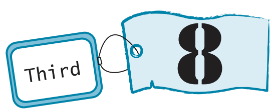


注意这里的做法。在交互模式中，只需键入变量名就可以显示这个变量的值，而不必使用 print。（不过程序中可不行。）

在这个例子中，并没有在 print 指令中求和，而是先取 First 的值和 Second 的值，将二者相加，创建一个新的值，名为 Third。Third 是 First 和 Second 的和。

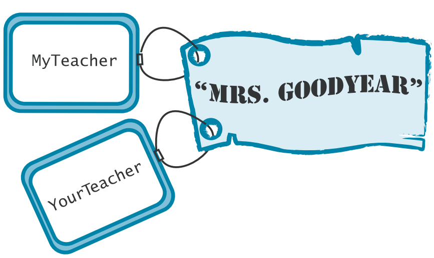






同一个东西可以有多个名字，可以在交互模式中试试这个指令：

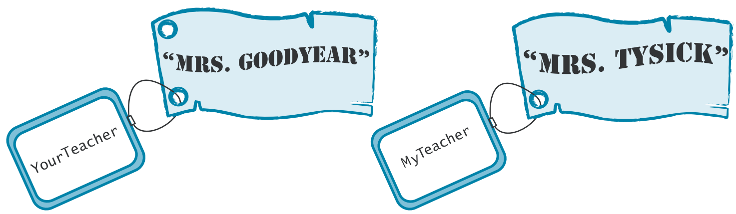
```
>>> MyTeacher = "Mrs. Goodyear"
>>> YourTeacher = MyTeacher
>>> MyTeacher
"Mrs. Goodyear"
>>> YourTeacher
"Mrs. Goodyear"
```



这就像在同一个东西上贴两个标签。一个标签写着 YourTeacher，另一个标签写着 MyTeacher，不过它们都贴在 "Mrs. Goodyear" 上。



卡特，这个问题问得好。答案是：不会。实际上，修改这个名字会创建一个新的值 "Mrs.Tysick"。标签 MyTeacher 会从 "Mrs.Goodyear" 上撕掉，贴到 "Mrs.Tysick" 上。你仍然有两个不同的名字（两个标签），不过，现在它们分别贴在两个不同的东西上，而不再贴在同一个东西上了。



2.3 名字里是什么

可以给变量取你喜欢的任何名字（严格地说，应该是几乎任何名字）。名字长短由你来定，里面可以有字母和数字，还可以有下划线（_）。

不过对于变量名还有几条规则。最重要的一点是名字是区分大小写的，即大写和小写是不同的。所以 teacher 和 TEACHER 是两个完全不同的名字。同样，first 和 First 也不相同。

另一条规则是变量名必须以字母或下划线字符开头。不能以数字开头，所以 4fun 不能作为变量名。

还有一条规则，变量名中不能包含空格。

如果你想知道 Python 中有关变量名的所有规则，可以查看本书后面的附录 A。

从前的美好时光



在一些较早的编程语言中，变量名的长度只能是一个字母，而且有些计算机只有大写字母，这说明变量名只有 26 种选择：A～Z！如果程序中需要的变量超过 26 个，那就难办了！

2.4　数字和字符串

目前为止，我们已经为字母（文本）和数字创建了变量。不过，在前面的加法例子中，Python 怎么知道我们指的是数字 5 和 3，而不是字符“5”和“3”呢？就像前面这句话一样，正是引号带来了差别。

字符或字符序列（字母、数字或标点符号）称为一个字符串（string）。要告诉 Python 你在创建一个字符串，就要在字符两边加上引号。至于使用单引号还是双引号，Python 并不太挑剔。单引号和双引号都是可以的：

```
>>> teacher = "Mr. Morton"   ← 双引号

>>> teacher = 'Mr. Morton'   ← 单引号
```

不过，字符串的开头和结尾必须使用同种类型的引号（要么都是双引号，要么都是单引号）。

如果键入一个数字而没有加引号，Python 就会知道这表示数值，而不是字符。可以试试看二者的区别：

```
>>> first = 5
>>> second = 3
>>> first + second
8
>>> first = '5'
>>> second = '3'
>>> first + second
'53'
```

没有引号时，5 和 3 都处理为数字，所以我们会得到二者的和。有引号时，'5' 和 '3' 处理为字符串，所以会得到两个字符“相加”的结果，也就是 '53'。还可以把由字母构成的字符串“加”在一起，第 1 章中已经见过这样的例子：

```
>>> print "cat" + "dog"
catdog
```

要注意，像这样将两个字符串相加时，它们之间没有空格。两个字符串会紧紧地拼接在一起。

注意！这个词有意思！

拼接

谈到字符串时我们说把它们“相加”（刚才就这么说过），不过这并不完全正确。把字符或字符串放在一起构成更长的字符串时，有一个特殊的称呼。并不是“相加”（相加只适用于数字），而是称为拼接（concatenation）。

我们会说拼接两个字符串。

长字符串

如果希望得到一个跨多行的字符串，必须使用一种特殊的字符串，称为三重引号字符串（triple-quoted string），就像下面这样：

```
long_string = """Sing a song of sixpence, a pocket full of rye,
Four and twenty blackbirds baked in a pie.
When the pie was opened the birds began to sing.
Wasn't that a dainty dish to set before the king?"""
```

这种字符串以 3 个引号开头和结尾。所用的引号可以是双引号也可以是单引号，所以也可以写成下面的形式：

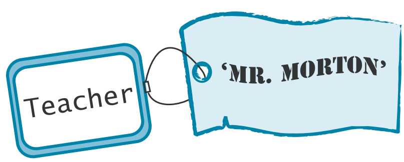
```
long_string = '''Sing a song of sixpence, a pocket full of rye,
Four and twenty blackbirds baked in a pie.
When the pie was opened the birds began to sing.
Wasn't that a dainty dish to set before the king?'''
```

如果希望多行文本显示在一起，而且你不希望每一行都使用一个单独的字符串，在这种情况下，三重引号字符串就非常有用。

2.5 它们有多“可变”

变量之所以叫做“变量”是有原因的，就是因为它们是……怎么说呢……是可变的！这是指你可以改变赋给它们的值。在 Python 中，这就要创建一个与原先不同的新东西，并把旧标签（名字）贴到这个新东西上。上一节中我们就采用这种方式改变了 MyTeacher。我们将标签 MyTeacher 从 "Mrs. Goodyear" 上取下来，把它贴到一个新东西 "Mrs. Tysick" 上。这样就为 MyTeacher 赋了一个新值。

下面再来试一个例子。还记得之前创建的变量 Teacher 吗？嗯，如果你还没有关闭 IDLE，这个变量就还在。可以检查看看：



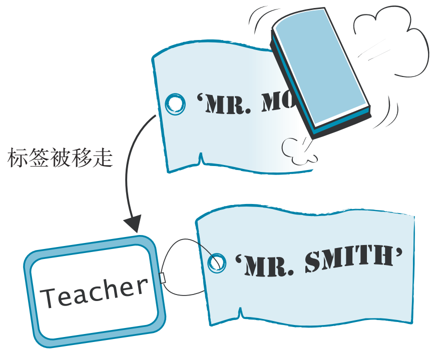

```
>>> Teacher
'Mr. Morton'
```

没错，确实还在。不过现在可以把它改成其他内容：

```
>>> Teacher = 'Mr. Smith'
>>> Teacher
'Mr. Smith'
```

我们创建了一个新东西 "Mr. Smith"，并把它命名为 Teacher。我们的标签从原来的值上取下来，贴到了这个新东西上。不过原来的 "Mr. Morton" 怎么样了呢？

应该记得，一个东西可以有多个名字（上面可以贴多个标签）。如果 "Mr. Morton" 上还有另一个标签，那么它还在计算机的内存里。不过，如果它上面再没有任何标签了，Python 就会发现再没有人需要它了，所以会把它从内存中删除。



这样一来，内存中就不会塞满那些没人用的东西。Python 会自动完成所有这些清理工作，根本不用你操心。

还有一点很重要，这里并没有真的把 "Mr. Morton" 改成 "Mr. Smith"。我们只是把标签从一个东西移到另一个东西上（重新指派名字）。Python 中有些东西（如数字和字符串）是不能改变的。你可以把它们的名字重新指派到其他东西上（就像我们刚才所做的一样），但是并不能对原先的东西做任何改变。

Python 中还有一些东西是可以改变的。第 12 章介绍列表（list）时我们会更多地讨论这方面的内容。

2.6　全新的我

还可以创建一个等于自己的变量：

```
>>> Score = 7
>>> Score = Score
```

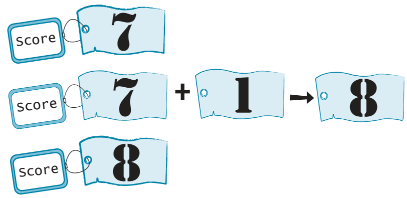
我敢打赌，你肯定在想："什么嘛，这一点儿用都没有！"你的想法没错。这实际上就是在说"我是我"。不过，稍稍做点改变，你就能成为一个全新的你！试试看：

```
>>> Score = Score + 1
>>> print Score
8
```

把 Score 从 7 改为 8

这里发生了什么？在第一行中，Score 标签本来贴在值 7 上。我们创建了一个新东西：Score + 1，也就是 7 + 1。这个新东西是 8。然后把 Score 标签从原来的东西（7）上取下来，贴到这个新东西（8）上。所以 Score 从 7 重新指派到 8。

要让变量等于某个东西，这个变量总会出现在等号（=）左边。巧妙的是，变量也可以出现在等号右边。这很有用，在很多程序中都会看到。最常见的用法是让变量自增（increment），也就是让它增加某个量（就像前面所做的），或者与之相反，也可以让变量自减（decrement），让它减少某个量。



- ❒ 开始时 Score = 7。
- ❒ 让它增加 1（得到 8），创建一个新东西。
- ❒ 把名字 Score 赋给这个新东西。

这样一来，Score 就从 7 变成了 8。

关于变量，有几个重要的问题必须记住。

- ❒ 程序可以在任何时间对变量重新赋值（把标签贴在新东西上）。这一点很重要，必须记住，因为编程中最常见的 bug 就是改变了不该改变的变量，或者尽管改变的变量无误，但是时机不合适。

  要避免这种情况，有效的方法是使用容易记的变量名。我们可能用过下面这两个变量名：

  ```
  t = 'Mr. Morton'
  ```

  或

  ```
  x1796vc47blahblah = 'Mr. Morton'
  ```

  不过这样在程序中会很难记住。如果使用这些变量名，出错的可能性会更大。应该尽量使用能够说明用途的名字，可以告诉你变量要用来做什么。

- ❒ 变量名区分大小写。这说明大写和小写是不同的。所以 teacher 和 Teacher 是两个完全不同的名字。

记住，如果想了解 Python 的所有变量命名规则，可以查看附录 A。



像程序员一样思考

我们曾经说过，你可以为变量取任何名字（不过前提是要满足命名规则），这一点不假。你可以把变量叫做 teacher 或者 Teacher，这两个名字都是可以的。

专业的 Python 程序员给变量命名时几乎总是以小写字母开头，其他计算机语言可能会采用不同风格。是否遵循 Python 风格由你来决定。因为我们使用的是 Python，所以这本书后面都会遵循这种风格。

你学到了什么

这一章中，你学到了以下内容。

- 如何使用变量在计算机内存中“记住”或保存信息。
- 变量也叫做“名字”或“变量名”。
- 变量可以是不同类型的东西，如数字和字符串。

测试题

1. 如何告诉 Python 变量是字符串（字符）而不是数字？
2. 一旦创建一个变量，能不能改变赋给这个变量的值？
3. 变量名 `TEACHER` 与 `TEACHEr` 相同吗？
4. 对 Python 来说，`'Blah'` 与 `"Blah"` 一样吗？
5. 对 Python 来说，`'4'` 是不是等同于 4 ？
6. 下面哪个变量名不正确？为什么？
   (a) `Teacher2`
   (b) `2Teacher`
   (c) `teacher_25`
   (d) `TeaCher`
7. `"10"` 是数字还是字符串？

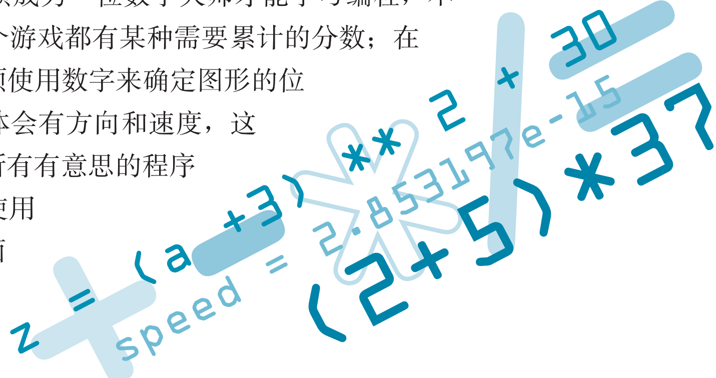
动手试一试

1. 创建一个变量，并给它赋一个数值（任何数值都行）。然后使用 `print` 显示这个变量。
2. 改变这个变量，可以用一个新值替换原来的值，或者将原来的值增加某个量。使用 `print` 显示这个新值。
3. 创建另一个变量，并赋给它一个字符串（某个文本）。然后使用 `print` 显示这个变量。
4. 像上一章一样，在交互模式中让 Python 计算一周有多少分钟。不过，这一次要使用变量。以 `DaysPerWeek`（每周天数）、`HoursPerDay`（每天小时数）和 `MinutesPerHour`（每小时分钟数）为名分别创建变量（或者也可以用自己取的变量名），然后将它们相乘。
5. 人们总是说没有足够的时间做到尽善尽美。如果一天有 26 个小时，那么一周会有多少分钟呢？（提示：改变 `HoursPerDay` 变量。）

第 3 章

基本数学运算

刚开始在交互模式中使用 Python 时，我们已经看到它可以完成简单的算术运算。现在来看 Python 还能对数字做些什么，还能完成哪些数学运算。也许你没有意识到，不过要知道，数学确实无处不在！特别是在编程中，我们一直都在使用数学。这并不是说你必须成为一位数学大师才能学习编程，不过可以想想看……每个游戏都有某种需要累计的分数；在屏幕上绘制图形时必须使用数字来确定图形的位置和颜色；移动的物体会有方向和速度，这都要用数字来描述。所有有意思的程序几乎都会以某种方式使用数字和数学。所以下面就来学习 Python 中有关数学和数字的一些基础知识。





顺便说一句，这里学习的很多知识同样适用于其他编程语言，也可以在电子表格之类的其他程序中使用。并不是只有 Python 采用这种方式完成数学运算。

3.1　四大基本运算

在第 1 章中我们已经看到 Python 可以做一些数学运算：使用加号（+）完成加法，另外使用星号（*）完成乘法。

如你所料，Python 使用连字号（–）（也称为减号）来做减法：

```
>>> print 8 - 5
3
```

由于计算机键盘上没有除号（÷），所以所有程序都使用前斜杠（/）表示除法。

```
>>> print 6/2
3
```

这是对的。不过有时 Python 做除法时会得到意外的结果：

```
>>> print 3/2
1
```

咦？我还以为计算机精通数学计算呢，原来不过如此！所有人都知道

```
3 / 2 = 1.5
```

这到底怎么回事？

嗯，虽然看起来好像很傻，其实 Python 确实想表现得聪明一些。要解释这个问题，你要知道整数和小数。如果你还不知道它们的区别，先来看看术语箱中简单的解释。

术语箱

整数（integer）就是我们平常数数时所说的数，如 1、2、3，另外还包括 0 和负数，如 −1、−2、−3。

小数（decimal number）也称为实数（real number），这些数有小数点而且后面有小数位，如 1.25、0.3752 和 −101.2。

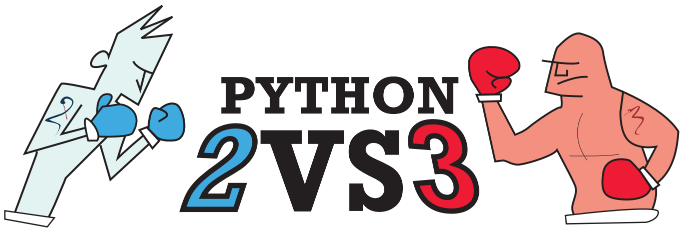
在计算机编程中，小数也称为浮点数（floating-point number，有时简写为 floats，或者如果只有一个浮点数，就简写为 float）。这是因为小数点会“浮动”。0.00123456 或 12345.6 都是浮点数。

因为你输入的 3 和 2 都是整数，所以 Python 认为你同样想要整数作为答案。所以它会把答案 1.5 取整为比它小且最接近的整数，也就是 1。换句话说，Python 完成了不带余数的除法。

要解决这个问题，可以这样试试看：

```
>>> print 3.0 / 2
1.5
```

这样就好多了！如果把两个数中的任何一个作为小数输入，Python 就会知道你想在答案中保留小数部分。



整除

在 Python 2 中，做除法的方式称为“整除”（floor division）。但 Python 3 的工作方式不一样，在 Python 3 中，如果你使用常规的除法操作符（前斜杠“/”），则做的是常规除法，而不是整除。

```
>>> print 3/2
1.5
```

要在 Python 3 中做整除，需要使用两个前斜杠（//）：

```
>>> print 3//2
1
```

这是 Python 2 和 Python 3 最显著的区别之一，也是导致很多 Python 2 程序与 Python 3 程序不兼容的原因。

切记！

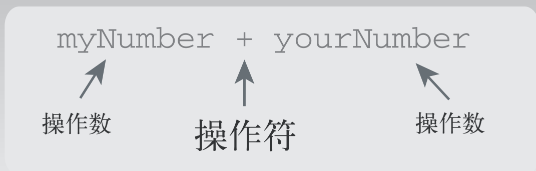
要记住 Python 2 的这种整除行为。这很重要，很多 Python 程序员（包括我自己）都曾经因为忘记这一点犯过错。

3.2 操作符

+、–、* 和 / 符号都称为操作符。这是因为它们会“操作”或处理放在符号两边的数字。= 号也是一个操作符，这称为赋值操作符（assignment operator），因为我们用它为一个变量赋值。

注意！这个词有意思！

操作符（operator）就是会对它两边的东西有影响或者有“操作”的符号。这种影响可能是赋值、检查或者改变一个或多个这样的东西。



完成算术运算的 +、–、* 和 / 符号都是操作符。

所操作的东西称为操作数（operand）。



3.3 运算顺序

下面哪一个正确？

```
2 + 3 * 4 = 20
```

还是

```
2 + 3 * 4 = 14
```

这要看你采用什么顺序来计算。如果先做加法，会得到

```
2 + 3 = 5，
```

然后得到

```
5 * 4 = 20
```

如果先做乘法，就会得到

```
3 * 4 = 12,
```

然后是

```
2 + 12 = 14
```

第二个顺序是正确的，所以正确答案是 14。在数学中有一种运算顺序（order of operation），指定了先计算哪些操作符，后计算哪些操作符，而不管它们的书写顺序如何。

在我们的这个例子中，尽管 + 号在 * 号前面，但是应当先算乘法。Python 会遵循正确的数学规则，所以它会先做乘法再做加法。可以在交互模式中试试看，看看能不能得到这个结果：

```
>>> print 2 + 3 * 4
14
```



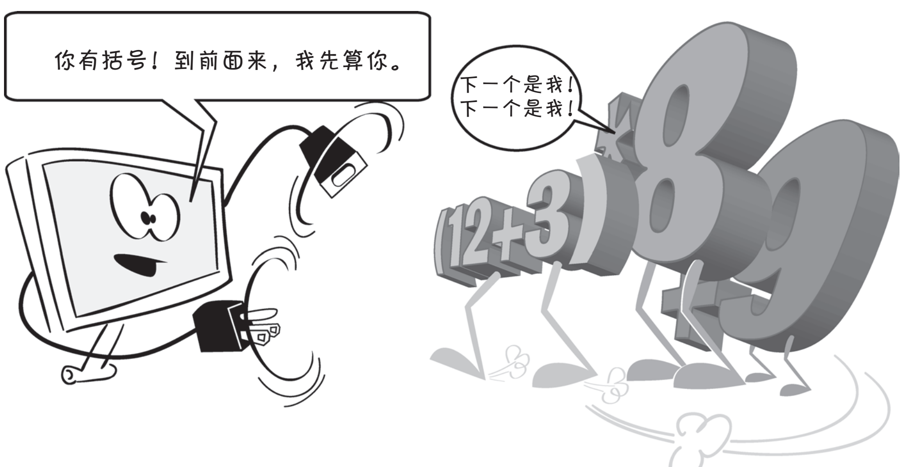
Python 使用的顺序与你在数学课上学到的（或者将要学到的）规则完全相同。指数运算最优先，然后是乘除，再后面是加减运算。

如果希望改变运算顺序，先完成某个运算，只需要在它两边加上括号（圆括号），比如：

```
>>> print (2 + 3) * 4
20
```

这一次，Python 会先做 2 + 3（因为有括号），可以得到 5，然后再做乘法 5 * 4，得到 20。

再强调一次，这与数学课上讲的是一样的。Python（和所有其他编程语言）都会遵循正确的数学规则和运算顺序。



3.4　另外两个操作符

还有两个算术操作符要告诉你。程序中需要的 99% 的操作符就是这两个操作符再加上前面刚讲的 4 个基本操作符。

指数——自乘为一个幂

如果把 3 乘 5 次，可以写成

```
>>> print 3 * 3 * 3 * 3 * 3
243
```

不过，这就等同于 3^5，或者“3 的指数为 5”，也就是“3 的 5 次幂”。Python 用一个双星号表示指数或者将一个数自乘为一个幂。

```
>>> print 3 ** 5
243
```

切记！

很多语言和程序可能使用其他符号来表示自乘为一个幂。一种常用的符号是 ^（例如 3^5）。如果在 Python 中使用这个符号，你不会得到一个错误消息，只不过答案不正确。（这是因为，^ 在 Python 中另有含义——我们可不希望这样！）这个问题可能很难调试。一定要使用 ** 操作符表示自乘为一个幂 [也称为求幂（exponentiation）]。

之所以使用指数而不是直接做多次乘法，这是因为键入时会更容易一些。不过更重要的原因是，利用 ** 还可以用非整数作为指数，如下：

```
>>> print 3 ** 5.5
420.888346239
```

要想利用乘法来做到这一点可不容易。

取余——求余数

在 Python 中第一次尝试除法时，我们已经看到，如果将两个整数相除，Python 2

会给你一个整数答案。(Python 3 使用 // 操作符。)也就是说，它在完成整数除法。不过，在整数除法中，答案实际上有两部分。

还记得刚开始学除法吗？如果两个数不能整除，最后会得到一个余数（remainder）：

7 / 2 = 3，余数是 1

7 / 2 的答案中有一个商（quotient），在这里就是 3，还有一个余数（remainder），这里的余数是 1。如果在 Python 中将两个整数相除，它会给你商。不过余数呢？

Python 有一个特殊的操作符来计算整数相除的余数。这称为取余（modulus）操作符，这个符号是百分号（%）。可以像这样使用：

```
>>> print 7 % 2
1
```

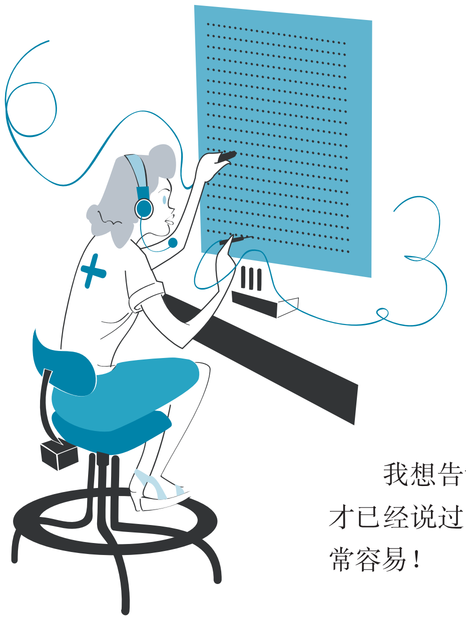
所以如果同时使用 / 和 %，就可以得到整数相除的完整答案：

```
>>> print 7 / 2
3
>>> print 7 % 2
1
```

可以看到，7 除以 2 得 3，余数是 1。如果做浮点数除法，会得到小数答案：

```
>>> print 7.0 / 2
3.5
```





实际上，既然你提到了这一点，应该说操作符和操作员确实很接近……就像老式电话接线员连接电话一样，算术操作符按同样的方式把数字连接在一起。

我想告诉你的还有另外两个操作符。我知道，我刚才已经说过只再讲两个，不过别担心，这两个操作符非常容易！

自增和自减

还记得上一章中的例子：`score = score + 1` 吗？我们说过，这称为自增（incrementing）。与它类似的是 `score = score - 1`，这称为自减（decrementing）。这些运算在编程中经常出现，因此有自己专门的操作符：+=（自增）和 -=（自减）。

可以像这样使用：

```
>>> number = 7
>>> number += 1
>>> print number
8
```

number 增 1

或者：

```
>>> number = 7
>>> number -= 1
>>> print number
6
```

number 减 1

其中第一个例子将 number 增 1（这会从 7 变成 8）。第二个例子将 number 减去 1（从 7 变成 6）。

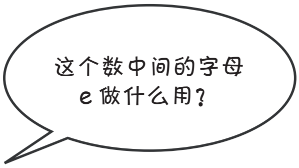
3.5　非常大和非常小

还记得第 1 章中将两个非常大的数相乘吗？我们得到的答案也是一个非常大的数。

有时 Python 会用一种稍微不同的方式显示非常大的数。可以在交互模式中试试看：

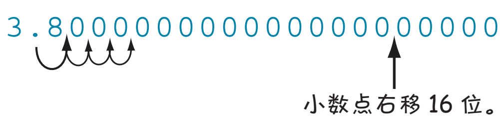
```
>>> print 9938712345656.34 * 4823459023067.456
4.79389717413e+025
>>>
```

（具体键入什么数并不重要——任何包含小数的大数值都可以。）

这个 e 是计算机中显示非常大或非常小的数时采用的一种方法。这叫做 E 记法（E-notation）。处理非常大（或非常小）的数时，要把所有数字以及小数位都显示出来可能很费劲。这种数在数学和科学领域经常出现。例如，如果一个天文程序要显示从地球到 Alpha Centaur 星的公里数，可能会显示为 38000000000000000 或者 38 000 000 000 000 000（38 后面有 15 个 0）。不论哪种方式，数完所有这些 0 都会让你累得够呛。

显示这些数还有另一种方式，就是使用科学计数法（scientific notation），就是一个小数再乘以一个 10 的幂。在科学计数法中，地球到 Alpha Centauri 的距离可以写作：3.8×10^{16}（看到了吗，16 抬高了，而且要小一点）。这读作“3.8 乘以 10 的 16 次幂”或者“3.8 乘以 10 的 16 次方”。它的意思就是，把 3.8 的小数点向右移 16 位，并在这个过程中根据需要补 0。



38000000000000000.0 = 3.8×10^{16}

如果可以像这里一样，把 16 写作指数，稍稍抬高一点，再写得小一点，科学计数法就很适用。如果你用纸和笔，或者使用一个支持上标的程序，就可以用科学计数法。

术语箱

上标（superscript）是指一个字符或一组字符比其余文本高一些，例如 10^{13}。这里的 13 就是上标。通常上标还要比正文小一点。

下标（subscript）也类似，不过这些字符会比其余文本低，同样也会小一点，比如 $\log_2$。这里的 2 就是一个下标。

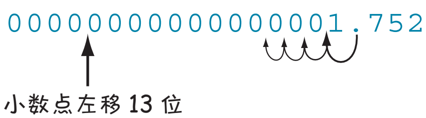
不过并不是哪里都能使用上标，所以还有另一种方法，就是 E 记法。E 记法只是科学计数法的另一种写法。

E 记法

在 E 记法中，这个数要写作 3.8E16 或者 3.8e16。读作“3.8 指数 16”或者简读作“3.8 e 16”。这里假设指数是 10 的幂。这就等同于写成 3.8×10^{16}。

在大多数程序和计算机语言（包括 Python）中，大写和小写 E 都是允许的。

对于非常小的数，如 0.0000000000001752，可以使用一个负指数。科学计数法会写作 1.752×10^{-13}，E 记法会写作 1.752e–13。负指数表示要把小数点向左移而不是向右移。



0.0000000000001752 = 1.752e-13

采用 E 记法，可以在 Python 中输入非常大和非常小的数（或者可以是任何数）。后面我们还会学习如何让 Python 使用 E 记法打印数。

试试采用 E 记法输入一些数：

```
>>> a = 2.5e6
>>> b = 1.2e7
>>> print a + b
14500000.0
```

尽管我们用 E 记法输入了数，但得出的答案却是一个常规的小数。这是因为，除非你特别要求，或者数字确实非常大或非常小（有很多个 0），否则 Python 不会用 E 记法显示数字。

可以试试看：

```
>>> c = 2.6e75
>>> d = 1.2e74
>>> print c + d
2.72e+75
```

这一次 Python 会自动用 E 记法显示答案，因为显示一个有 73 个 0 的数太不可思议了！

如果希望用 E 记法显示类似 14 500 000 的数，需要给 Python 下达一些特殊的指令。我们将在本书的第 21 章学习更多相关内容。

别担心，放松点！



如果你还不太理解 E 记法到底是怎么回事，不用担心。这本书后面的程序中不会用到它。我只是想让你了解一下它的原理，没准以后你会用到。

如果使用 Python 来完成一些数学运算，得到的答案是一个类似 5.673745e16 的数，至少现在你知道这是一个非常大的数，而不是出现了什么错误。

指数与 E 记法

不要把自乘得到幂（也称为求幂）和 E 记法弄混了。

- 3**5 表示 3^5，或“3 的 5 次幂”，也就是 3 * 3 * 3 * 3 * 3，等于 243。
- 3e5 表示 3 * 10^5 或者“3 乘以 10 的 5 次幂”，也就是 3 * 10 * 10 * 10 * 10 *10，结果等于 300 000。
- 求幂是指一个数自乘得到幂。E 记法表示乘以 10 的几次幂。

有些人可能会把 3e5 和 3**5 都读作“3 指数 5”，不过，它们是完全不同的。怎么读并不重要，只要你懂得它们分别代表什么含义。

你学到了什么

在这一章，你学到了以下内容。

- 用 Python 如何完成基本数学运算。
- 整数和浮点数。
- 求幂（自乘得到一个幂）。
- 如何计算取余（余数）。
- E 记法的有关内容。

测试题

1. Python 中乘法使用哪个符号？
2. Python 计算 8 / 3 的答案是什么？
3. 怎么得到 8 / 3 的余数？
4. 怎么得到 8 / 3 的小数结果？
5. Python 中计算 6 * 6 * 6 * 6 的另一种做法是什么？
6. 采用 E 记法，17 000 000 要写作什么？
7. 4.56e–5 如果按常规的写法是什么（不是 E 记法）？

动手试一试

1. 使用交互模式或者编写一个小程序解决下面的问题。
   (a) 3 个人在餐厅吃饭，想分摊饭费。总共花费 35.27 美元，他们还想留 15% 的小费。每个人该怎么付钱？
   (b) 计算一个 12.5m×16.7m 的矩形房间的面积和周长。
2. 写一个程序，把温度从华氏度转换为摄氏度。转换公式是 C = 5 / 9* (F–32)。（提示：当心整除问题！）
3. 你知道怎么计算坐车去某个地方需要花多长时间吗？相应的公式（用文字表述）是“旅行时间等于距离除以速度”。编写一个程序，计算以 80 km/h 的速度行驶 200 km 需要花多长时间，并显示答案。

第 4 章

数据的类型

我们已经看到，至少可以为一个变量赋 3 种不同类型的值（保存在计算机内存中）：整数、浮点数和字符串。Python 还有其他一些数据类型，后面将会学到，不过对现在来说，这 3 个类型就足够了。这一章中，我们将学习怎样区分一个值究竟是什么类型。还会了解如何由一个类型建立另一个类型。

4.1 改变类型

很多情况下，我们需要将数据从一种类型转换成另一种类型。例如，想要打印一个数字时，就需要把它转换成文本，使它能够出现在屏幕上。Python 的 `print` 命令可以为我们实现这点。不过，有时我们只是想转换而不需要打印出来，或者需要从字符串转换成数字（这是 `print` 无法做到的）。这称为类型转换（type conversion）。这该如何做到呢？

Python 实际上并没有把一个东西从一种类型“转换”成另一种类型。它只是由原来的东西创建一个新东西，而且这个新东西正是你想要的类型。下面给出一些函数，它们可以把数据从一种类型转换为另一种类型。

- `float()` 从一个字符串或整数创建一个新的浮点数（小数）。
- `int()` 从一个字符串或浮点数创建一个新的整数。
- `str()` 从一个数（可以是任何其他类型）创建一个新的字符串。

`float()`、`int()` 和 `str()` 后面有小括号，因为它们不是 Python 关键字（如 `print`）——它们只是 Python 的内置函数（function）。

后面我们还会学习更多有关函数的内容。现在只需要知道：可以把你想要转

换的值放在函数后面的小括号里。要说明这一点，最好的办法就是举一些例子。在 IDLE shell 中，采用交互模式完成下面的例子。

将整数转换为浮点数

下面先从整数开始，由它创建一个新的浮点数（小数），这里要使用 float()：

```
>>> a = 24
>>> b = float(a)
>>> a
24
>>> b
24.0
```

注意 b 得到一个小数，末尾有一个 0。这就告诉我们这是一个浮点数而不是整数。变量 a 保持不变，因为 float() 不会改变原来的值——它只是创建一个新的值。

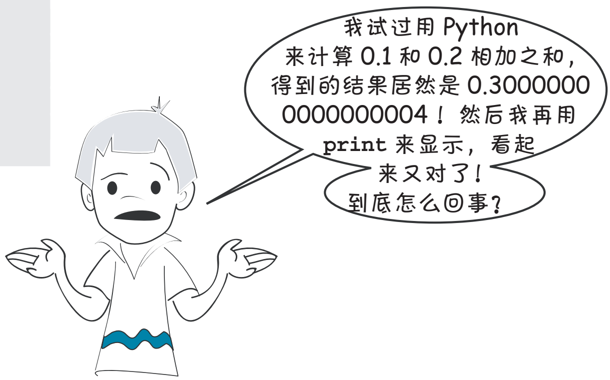
要记住，在交互模式中，可以直接键入变量名（而不需要使用 print），Python 会显示这个变量的值（这在第 2 章中曾经见过）。不过这只在交互模式中奏效，在程序中是行不通的。

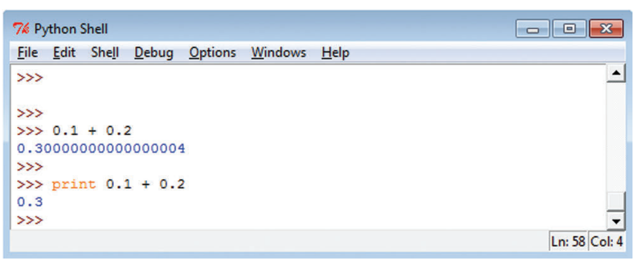
将浮点数转换为整数

下面再反过来试试，从一个小数用 int() 创建一个整数：

```
>>> c = 38.0
>>> d = int(c)
>>> c
38.0
>>> d
38
```

我们创建了一个新的整数 d，这是 c 的整数部分。





是吗？怎么会发生这种事情？卡特，我想肯定是你的计算机发疯了！

当然这只是开玩笑。实际上，这个问题有一个解释，你可以看看下面的“到底怎么回事？”。

到底怎么回事？



还记得我们说过计算机在内部都使用二进制吧？对，Python 存储的所有数都是作为二进制存储的。对于 0.1 和 0.2 之和，Python 会用足够多的二进制位（比特）创建一个浮点数（小数）来保证 15 个小数位。不过这个二进制数并不完全等于 0.3，它只是相当相当接近。（在这里，误差是 0.000000000000004。）这个差称为舍入误差（roundoff error）。

在交互模式中键入 0.1+0.2 这个表达式时，Python 会显示它存储的原始数值，包括所有的小数位。使用 print 时，你会得到期望的结果，因为 print 更聪明一点，它很清楚要四舍五入显示 0.3。

这就像问一个人时间。他可能会说“12 点 44 分 53 秒”。不过大多数人都只是说“差一刻一点”，因为他们知道你不需要那么精确。所有计算机语言中浮点数都存在舍入误差。对于不同的计算机或者不同的语言来说，你得到的正确的位数可能有所不同，不过都会使用同样的基本方法来存储浮点数。

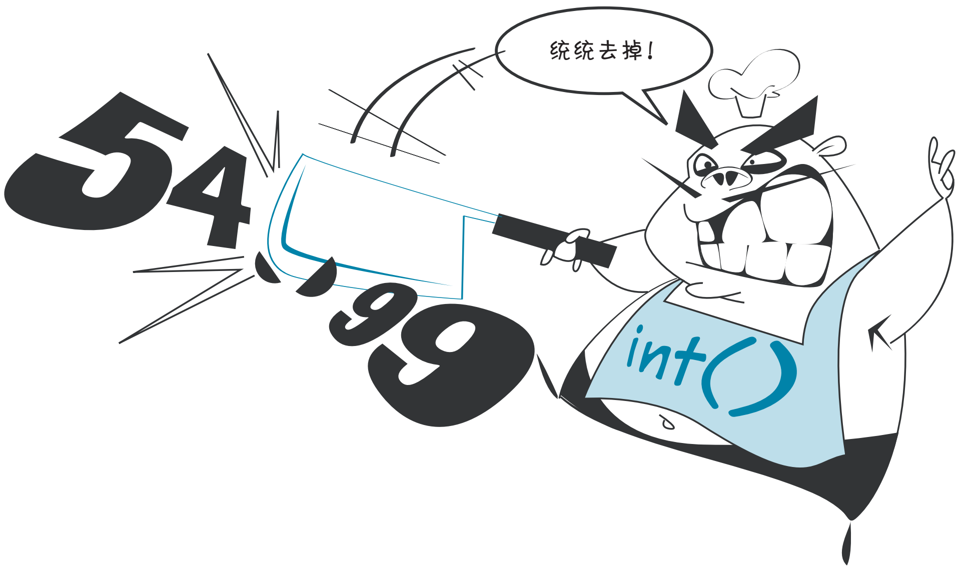
通常舍入误差都很小，所以不需要担心这些误差。

下面再试试另一个转换：

```
>>> e = 54.99
>>> f = int(e)
>>> print e
54.99
>>> print f
54
```

尽管 54.99 与 55 很接近，但是得到的整数仍然是 54。int() 函数总是下取整。它不会给你最接近的整数，而是会给出下一个最小的整数。实际上 int() 函数就是去掉小数部分。

如果想得到最接近的整数，也有一个办法。这个办法到第 21 章再告诉你。



将字符串转换为浮点数

还可以从字符串创建一个数，就像这样：

```
>>> a = '76.3'
>>> b = float(a)
>>> a
'76.3'
>>> b
76.3
```

注意，显示 a 时，结果两边有引号。Python 通过这种方式告诉我们 a 是一个字符串。显示 b 时，会得到浮点数值，这里包括所有小数位（就像卡特之前做的一样）。

4.2　得到更多信息：type()

上一节说过，我们通过看引号来确定一个值究竟是数还是字符串。要确定它是一个数还是字符串还有一种更直接的方法。

Python 还提供了函数 type()，它可以明确地告诉我们变量的类型。

下面试试看：

```
>>> a = '44.2'
>>> b = 44.2
>>> type(a)
<type 'str'>
>>> type(b)
<type 'float'>
```

type() 函数告诉我们 a 的类型是 'str'，这代表字符串（string），b 的类型是 'float'，很明白，不用猜也知道这代表浮点数！

4.3 类型转换错误

当然，如果向 int() 或 float() 提供的不是一个数，它就会不正常。

下面来试试看：

```
>>> print float('fred')
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print float ('fred')
ValueError: could not convert string to float: fred
```

我们得到了一个错误消息。这个非法文字（invalid literal）错误消息说明 Python 不知道怎么从 "fred" 创建一个数。如果是你，你知道吗？

4.4 使用类型转换

再来看第 3 章“动手试一试”中从华氏度到摄氏度的温度转换程序，应该记得当时需要修正整除行为才能得到正确的答案，需要把 5 改为 5.0 或者把 9 改成 9.0：

```
cel = 5.0 / 9 * (fahr - 32)
```

float() 函数给出了另一种做法：

```
cel = float(5) / 9 * (fahr - 32)
```

或

```
cel = 5 / float(9) * (fahr - 32)
```

可以试试看。

0011000111001110000110110100011011010111001100011001100110100110011001000110

你学到了什么

在这一章，你学到了以下内容。

- 完成类型转换（或者更准确地说，从某些类型创建另外一些类型）：str()、int() 和 float()。
- 直接显示值，而不使用 print。
- 使用 type() 查看变量的类型。
- 舍入误差及其出现的原因。

测试题

1. 使用 int() 将小数转换为整数，结果是上取整还是下取整？
2. 在温度转换程序中，可以这样做吗？

   ```
   cel = float(5 / 9 * (fahr - 32))
   ```

   这样呢？

   ```
   cel = 5 / 9 * float(fahr - 32)
   ```

   如果不行，为什么？
3. （挑战题）除了 int() 不使用任何其他函数，如何对一个数四舍五入而不是下取整？（例如，13.2 会下取整为 13，但是 13.7 会上取整为 14。）

动手试一试

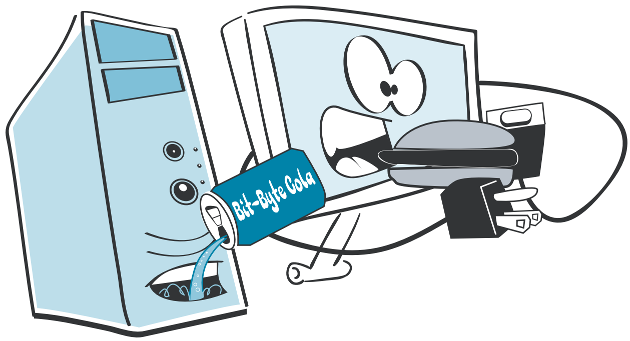
1. 使用 float() 从一个字符串（如 '12.34'）创建一个数。要保证结果确实是一个数！
2. 试着使用 int() 从一个小数（56.78）创建一个整数。答案是上取整还是下取整？
3. 试着使用 int() 从一个字符串创建整数。要保证结果确实是一个整数！

第 5 章
输　　入

到现在为止，希望程序“处理一些数”时，都必须把这些数直接放在代码中。例如，如果编写了第 3 章中的温度转换程序，你可能会把要转换的温度直接放在代码中。如果想要转换一个不同的温度，就必须修改代码。

如果你希望用户在程序运行时输入他想转换的温度呢？之前我们说过，一个程序有 3 大部分：输入、处理和输出。我们的第一个程序只有输出。温度转换程序有处理（转换温度）和输出，但是没有输入。现在该向程序增加第三个部分了：输入。输入就是指在程序运行时向其提供某样东西，也就是某种信息。

这样一来，我们就能写出与用户交互的程序，这就有趣多了。

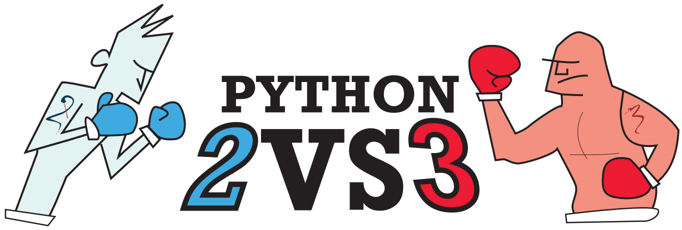


Python 有一个内置函数，名为 raw_input()，可以用这个函数从用户那里得到输入。在这一章中，我们将学习如何在程序中使用 raw_input()。

5.1 raw_input()

raw_input() 函数从用户那里得到一个字符串。正常情况下会从键盘得到这个输入，也就是说，用户要键入输入。

raw_input() 也是一个 Python 内置函数，就像 str()、int()、float() 和 type() 一样（在第 4 章中已经见过这些函数）。后面还会学习更多有关函数的内容。不过对现在来说，只需要记住使用 raw_input() 时要加上小括号（圆括号）。



Raw input

在 Python 3 中，raw_input() 改名为 input() 了。它与 Python 2 中的 raw_input() 完全一样。

可以这样来使用：

```
someName = raw_input()
```

这会让用户键入一个字符串，并把它赋给名字 someName。

现在把它放在程序里。在 IDLE 中创建一个新文件，键入代码清单 5-1 中的代码。

代码清单 5-1　使用 raw_input() 得到一个字符串

```
print "Enter your name: "
somebody = raw_input()
print "Hi", somebody, "how are you today?"
```

保存这个程序，并在 IDLE 中运行，看看它如何工作。应该可以看到类似下面的结果：

```
Enter your name:
Warren
Hi Warren how are you today?
```

我键入了我的名字，程序把它赋给了 somebody。

5.2　print 命令和逗号

通常情况下，希望从用户得到输入时，必须告诉他你想要什么，应当提供类似这样的一个消息：

```
print "Enter your name: "
```

然后用 raw_input() 函数得到用户的响应：

```
someName = raw_input()
```

如果运行这些代码行，并键入你的名字，会得到：

```
Enter your name:
Warren
```

如果希望用户在消息的同一行上键入他的答案，只需要在 print 语句的末尾放上一个逗号，就像这样：

```
print "Enter your name: ",
someName = raw_input()
```

注意逗号放在结束引号的外面。

如果运行这个代码，会得到：

```
Enter your name: Warren
```

逗号可以用来把多个 print 语句合并到同一行上。逗号只是表示“打印完这个内容后不要跳转到下一行”。代码清单 5-1 的最后一行就是这么做的。

在 IDLE 编辑器窗口中键入代码清单 5-2 中的代码，并运行这个程序。

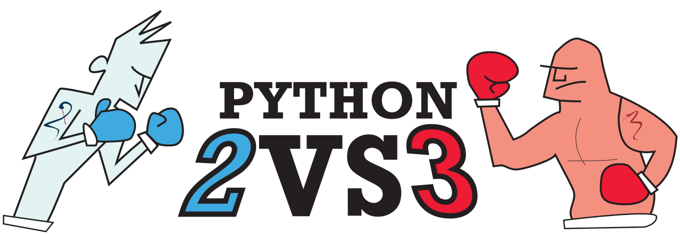
代码清单 5-2 逗号用来做什么？

```
print "My",
print "name",
print "is",
print "Dave."
```

运行这个程序时应该会得到这样的结果：

```
My name is Dave.
```

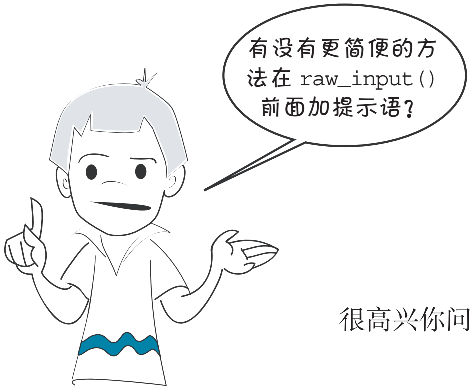
注意到了吗？引号中的每个词末尾都没有空格，但是运行这个程序时每个单词之间却出现了空格。使用逗号将多个 print 语句合并到同一行时，Python 会增加一个空格。



行尾逗号

在 Python 3 中，通过在行尾添加逗号使得打印的内容都在同一行的方法不再有效。而且，在 Python 3 中使用 print() 时，要打印的内容必须被包含在一对括号中。所以如果你在使用 Python 3，代码清单 5-2 应该是这样：

```
print( "My" , end="  ")
print( "name" , end="  ")
print( "is" , end="  ")
print( "Dave." , end="  ")
```



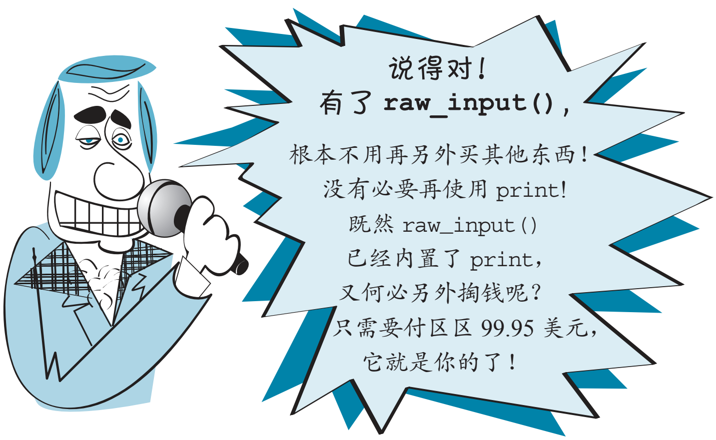
很高兴你问这个问题！我正要讲到这一点。

打印 raw_input() 提示语的简便方法

打印提示消息还有一种简便方法。raw_input() 函数可以直接打印消息，所以你根本不必使用 print 语句：

```
someName = raw_input ("Enter your name: ")
```

这就像 raw_input() 函数内置了 print 一样。从现在起我们都将使用这个简便方法。



5.3　输入数字

我们已经见过如何使用 raw_input() 来得到字符串。但是如果希望得到一个数该怎么做呢？毕竟，我们之所以讨论输入，原本就是为了让用户为我们的温度转换程序输入温度。

如果你读过第 4 章，应该已经知道答案了。可以从 raw_input() 给我们的字符串使用 int() 或 float() 函数创建一个数。可以像这样：

```
temp_string = raw_input()
fahrenheit = float(temp_string)
```

先使用 raw_input() 得到用户的输入（一个字符串）。然后使用 float() 由这个字符串创建一个数。得到温度（作为浮点数）后，为它指定名字 fahrenheit。

不过还有一种简便方法。只需一步就可以完成所有这些工作，如下：

```
fahrenheit = float(raw_input())
```

所做的工作是一样的。它由用户得到字符串，然后从这个字符串创建一个数。这里只是稍稍少写一点代码。

下面在我们的温度转换程序中使用这种方法。试着运行代码清单 5-3 中的程序，看看会得到什么。

代码清单 5-3 使用 raw_input() 转换温度

```
print "This program converts Fahrenheit to Celsius"
print "Type in a temperature in Fahrenheit: ",
fahrenheit = float(raw_input())          ← 使用 float(raw_input()) 从用户得到温度（华氏度）
celsius = (fahrenheit - 32) * 5.0 / 9
print "That is",
print celsius,                           注意这些代码行末尾的逗号
print "degrees Celsius"
```

还可以把代码清单 5-3 最后 3 行合并为一行，像这样：

```
print "That is", celsius, "degrees Celsius"
```

这实际上是之前 3 个 print 语句的简写形式。

结合 int() 使用 raw_input()

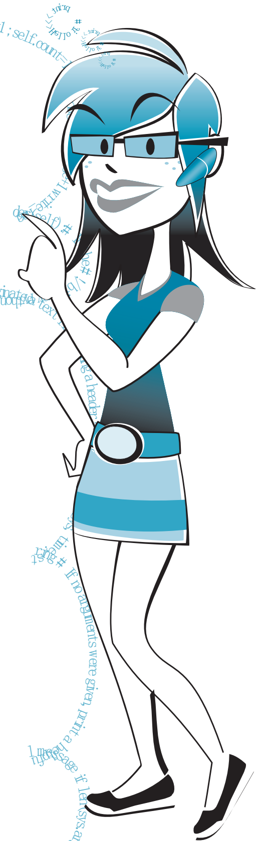
如果你希望用户输入的数总是整数（而不是小数），可以用 int() 来转换，例如：

```
response = raw_input("How many students are in your class: ")
numberOfStudents = int(response)
```

像（Python）程序员一样思考

得到数字输入还有一种方法。Python 2 有一个名叫 input() 的函数，可以直接提供一个数，所以不必使用 int() 或 float() 来转换。我们在第 1 章的猜数程序中用过这个函数，因为这是从用户得到一个数的最简单的方法。

但为了保持一致，我们在本书的剩余部分会始终使用 raw_input()。而且，Python 3 中去除了 input() 函数（可以直接获取数字而不需要进行转换），只有 raw_input()。更令人感到混乱的是，Python 2 中的 raw_input() 在 Python 3 中改名为 input()，但它的功能仍然和你在这一章中见到的这个函数一样，只会得到字符串。因为我们很清楚怎样从一个字符串创建一个数，所以建议使用 raw_input()，而不要用 Python 2 中的 input()。



5.4　来自互联网的输入

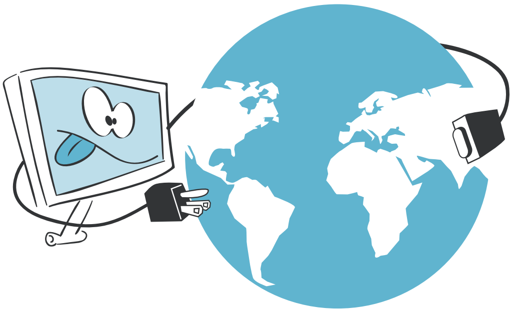
通常，程序的输入都来自用户。不过还有其他一些方法得到输入。可以从计算机硬盘上的文件中得到输入（这个内容会在第 22 章介绍），或者也可以从互联网获取输入。

如果你能连接互联网，可以试试代码清单 5-4 中的程序。它会从这本书的网站打开一个文件，为你显示这个文件中的消息。

代码清单 5-4　从互联网上的一个文件得到输入

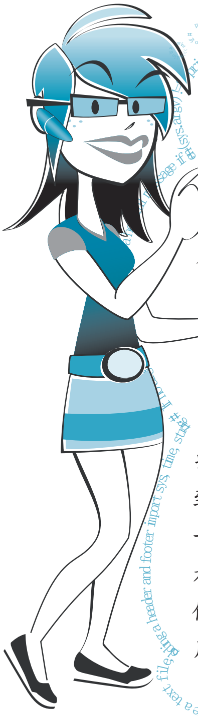
```
import urllib2
file = urllib2.urlopen('http://helloworldbook2.com/data/message.txt')
```

```
message = file.read()
print message
```



就这么简单。只需要区区 4 行代码，你的计算机就可以通过互联网得到这本书网站上的一个文件，并显示这个文件。如果试着运行这个程序（假设你的互联网连接工作正常），你会看到这个消息。

如果你在办公室或学校的计算机上尝试这个程序，很可能无法正常工作。这是因为，有些办公室和学校使用一种名叫代理的东西连接到互联网。代理就是另一台计算机，它相当于互联网与学校或办公室之间的一座桥梁或一条通路。取决于代理的设置，这个程序可能不知道如何通过代理连接到互联网。如果从家里的计算机（或者其他可以直接连接互联网而不需要通过代理的地方）运行这个程序，应该能正常工作。



像程序员一样思考

根据你使用的操作系统（Windows、Linux 或 Mac OS X），运行代码清单 5-4 中的程序时，你可能会在每行末尾看到小方块或类似 \r 的字符。这是因为，不同的操作系统使用不同的方法来指示文本行的结束。Windows（和之前的 MS-DOS）使用两个字符：CR（回车）和 LF（换行）来表示。Linux 只使用 LF，Mac OS X 只使用 CR。

有些程序可以处理所有这些情况，不过有些程序（比如 IDLE）看到行结束符与它期望的不一致时，就会不知所措。发生这种情况时，它们会显示一个小方块，表示“我不理解这个字符”。你可能会看到这样的小方块，也可能看不到，这取决于你在使用什么操作系统，还取决于你如何运行程序（使用 IDLE 还是采用另外某种方法）。

你学到了什么

在这一章，你学到了以下内容。

- 用 raw_input() 输入文本。
- 向 raw_input() 增加一个提示消息。
- 结合 int() 和 float() 使用 raw_input() 输入数字。
- 使用逗号将多行打印到一行上。

测试题

1. 对于下面这行代码：

```
answer = raw_input()
```

如果用户键入 12，answer 的数据类型是什么？是字符串还是一个数？

2. 怎么让 raw_input() 打印一个提示消息？

3. 怎么使用 `raw_input()` 得到一个整数？
4. 怎么使用 `raw_input()` 得到一个浮点数（小数）？

动手试一试

1. 在交互模式中建立两个变量，分别表示你的姓和名。然后使用一条 `print` 语句，把姓和名打印在一起。
2. 编写一个程序，先问你的姓，再问名，然后打印一条消息，在消息中包含你的姓和名。
3. 编写一个程序询问一间长方形房间的尺寸（单位是米），然后计算覆盖整个房间总共需要多少地毯，并显示出来。
4. 编写一个程序先完成第 3 题的要求，不过还要询问每平方尺地毯的价格。然后主程序显示下面 3 个内容：
   - 总共需要多少地毯，单位是平方米。
   - 总共需要多少地毯，单位是平方尺（1 平方米 = 9 平方尺）。
   - 地毯总价格。
5. 编写一个程序帮助用户统计她的零钱。程序要问下面的问题。
   - “有多少个五分币？”
   - “有多少个二分币？”
   - “有多少个一分币？”

   让程序给出这些零钱的总面值。

第 6 章

GUI——图形用户界面

到目前为止，我们的所有输入和输出都只是 IDLE 中的简单文本。不过现代计算机和程序会使用大量的图形。如果我们的程序中也有一些图形就太好了。在这一章中，我们会开始建立一些简单的 GUI。这说明从现在开始，我们的程序看上去就会像你平常熟悉的那些程序一样，将会有窗口、按钮之类的图形。

6.1 什么是 GUI

GUI 是 Graphical User Interface（图形用户界面）的缩写。在 GUI 中，并不只是键入文本和返回文本，用户可以看到窗口、按钮、文本框等图形，而且可以用鼠标点击，还可以通过键盘键入。我们目前为止完成的程序都是命令行或文本模式程序。GUI 是与程序交互的一种不同的方式。有 GUI 的程序仍然有 3 个基本要素：输入、处理和输出，但它们的输入和输出更丰富、更有趣一些。

顺便说一句，计算机上有 GUI 当然是可以的，但是要避免粘上黏性的东西哦。否则键盘将无法发挥效力，也会让键入很困难！



6.2　第一个 GUI

我们一直都在使用 GUI，实际上已经用过很多。Web 浏览器是 GUI，IDLE 也是 GUI。现在我们就来建立自己的 GUI。为了做到这一点，要从 EasyGui 寻求一些帮助。

EasyGui 是一个 Python 模块，利用这个模块可以很容易地建立简单的 GUI。我们还没有具体讨论过模块（第 15 章会介绍这方面的内容），不过应该知道：模块就是一种扩展方法，通过它可以向 Python 增加非内置的内容。

如果你使用这本书的安装程序来安装 Python，那么你已经安装了 EasyGui。否则，可以从 http://easygui.sourceforge.net/ 下载。

安装 EasyGui

可以下载 easygui.py 或者一个包含 easygui.py 的 zip 文件。要安装这个模块，只需要把文件 easygui.py 放在 Python 能找到的位置。这个位置是哪里呢？

Python 路径

Python 会在硬盘上的一组位置中查找可以使用的模块。这个工作可能有些复杂，因为在 Windows、Mac OS X 和 Linux 上，所查找的这组位置各不相同。不过，如果把 easygui.py 放在 Python 安装的位置中，Python 肯定能找到它。所以，要在你的硬盘上查找一个名叫 Python27 的文件夹，再把 easygui.py 放在这个文件夹里。

建立 GUI

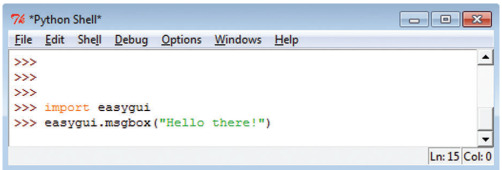
启动 IDLE，在交互模式键入以下命令：

```
>>> import easygui
```

这会告诉 Python 你打算使用 EasyGui 模块。如果没有得到错误消息，说明 Python 找到了 EasyGui 模块。如果收到一个错误消息，或者 EasyGui 看上去无效，可以访问本书网站（www.helloworldbook2.com），从中可以找到一些其他的帮助。

现在来建立一个包含 OK 按钮的简单消息框：

```
>>> easygui.msgbox("Hello There!")
```



EasyGui msgbox() 函数用于创建一个消息框。大多数情况下，EasyGui 函数的名就是相应英语单词的缩写。

使用 msgbox() 时，会看到类似这样的结果：



如果点击 OK 按钮，这个消息框会关闭。

6.3　GUI 输入

我们只看过一种 GUI 输出，就是一个消息框。不过输入呢？还可以使用 EasyGui 得到输入。

在交互模式中运行前面的例子时，你点击 OK 按钮了吗？如果点击了这个按钮，应该已经在 shell 或终端或命令窗口中见过这样的结果：

```
>>> import easygui
>>> easygui.msgbox("Hello there!")
'OK'
```

'OK' 部分就是 Python 和 EasyGui 在告诉你：用户点击了 OK 按钮。EasyGui 会返回信息来告诉你用户在 GUI 中做了什么：点击了什么按钮，键入了哪些内容等等。可以为这个响应指定一个名字（把它赋给一个变量）。试试看：

```
>>> user_response = easygui.msgbox("Hello there!")
```

在消息框中点击 OK 将它关闭。然后键入：

```
>>> print user_response
OK
```

现在用户的响应（OK）有了一个变量名 user_response。下面再来看其他几种使用 EasyGui 得到输入的方法。

我们刚才看到的消息框实际上只是对话框（dialog box）的一个例子。对话框包含一些 GUI 元素，用来告诉用户某些信息，或者从用户得到一些输入。输入可以是按钮点击（如 OK），或者文件名，也可以是某个文本（字符串）。

EasyGui msgbox 就是包含一条消息和一个 OK 按钮的对话框。不过还可以有不同类型的对话框，包含更多的按钮和其他内容。



6.4　选择你的口味

我们将举一个挑选冰淇淋口味的例子来学习利用 EasyGui 从用户得到输入（冰淇淋口味）的不同方法。

有多个按钮的对话框

下面来创建一个包含多个按钮的对话框（如消息框）。具体做法是使用一个按钮框（button box，buttonbox）。下面来建立一个程序，而不是在交互模式中完成。

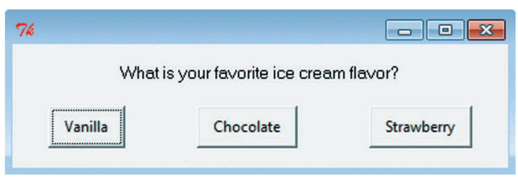
在 IDLE 中新建一个文件。键入代码清单 6-1 中的程序。

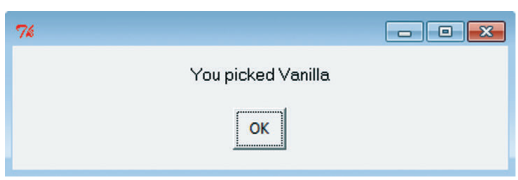
代码清单 6-1　使用按钮得到输入

```
import easygui
flavor = easygui.buttonbox("What is your favorite ice cream flavor?",
                  choices = ['Vanilla', 'Chocolate', 'Strawberry'] )  ← 选择列表

easygui.msgbox ("You picked " + flavor)
```

方括号中的代码称为一个列表（list）。我们还没有讨论列表，这部分内容将在第 12 章介绍。对现在来说，只需要键入这些代码，让这个 EasyGui 程序能够工作（如果你确实很好奇，也可以跳到第 12 章看个究竟……）。





保存文件（我的文件就命名为 ice_cream1.py），运行这个程序，你就会看到右边这个界面。

然后，根据你选择的口味，你会看到右图这样的结果了。

这是怎么做到的？用户点击的按钮的标签就是输入（input）。我们为这个输入指定了一个变量名，在这里就是 flavor。这就像使用 raw_input()，只不过用户并不是键入，而是点击一个按钮。这正是 GUI 的关键。

选择框

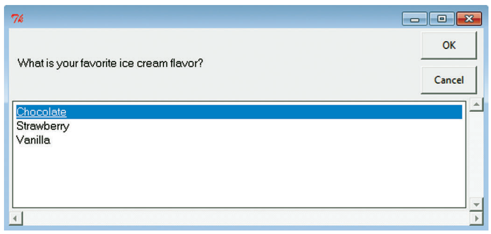
下面来看用户选择口味的另一种方法。EasyGui 提供了一种选择框（choice box，choicebox），它会显示一个选择列表。用户可以选择其中之一，然后点击 OK 按钮。

要尝试选择框，只需要对代码清单 6-1 中的程序做一个很小的修改：把 buttonbox 改为 choicebox。这个新版本的程序见代码清单 6-2。

代码清单 6-2　使用选择框得到输入

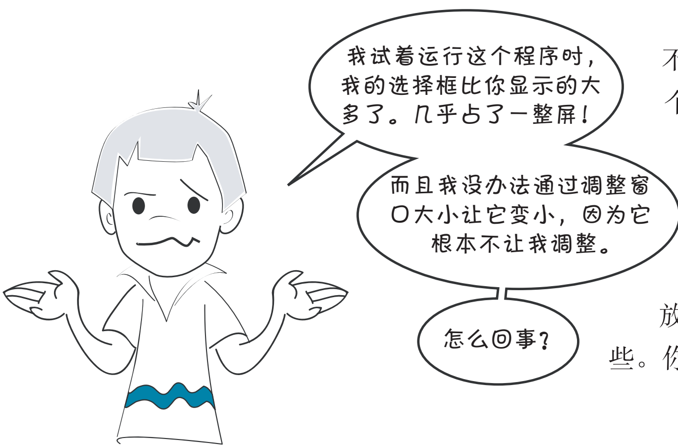
```
import easygui
flavor = easygui.choicebox("What is your favorite ice cream flavor?",
                  choices = ['Vanilla', 'Chocolate', 'Strawberry'] )
easygui.msgbox ("You picked " + flavor)
```

保存代码清单 6-2 中的程序并运行。你会看到类似右图的结果。



选择一个口味然后点击 OK 时，你会看到与前面相同的消息框。注意，除了用鼠标点击选择，还可以用键盘上的上下箭头键选择一个口味。

如果点击 Cancel，程序会结束，你还会看到一个错误。这是因为程序的最后一行希望得到某个文本（如 Vanilla），倘若你点击 Cancel，它将得不到任何输入。



我也遇到了同样的问题。不过因为在这本书里放上这个巨大的选择框不太合适，所以我要了点小聪明，稍稍做了点处理！我修改了 easygui.py，让选择框变小一些，这样放在这本书里看上去会好一些。你不需要这么做，但如果你

确实想试试看，下面我就把步骤告诉你。不过提醒你一句，这可有点复杂哦！

(1) 找出 easygui.py 文件中以 def__choicebox 开头的一节（在我的 easygui.py 中大约在 934 行）。要记住，大多数编辑器，都会在靠近窗口最下面的某个位置显示出代码行号。

(2) 从这个位置向下大约 30 行（大概是 970 行），会看到类似下面的代码行：

```
root_width = int((screen_width * 0.8))
root_height = int((screen_height * 0.5))
```

(3) 把 0.8 改为 0.4，再把 0.5 改成 0.25。保存对 easygui.py 做的这些修改。下一次运行程序时，选择框窗口就会小一些了。

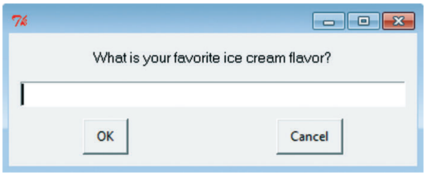
文本输入

这一章中的例子允许用户从你（程序员）提供的一组选项中做出选择。如果你想像 raw_input() 一样（也就是让用户键入文本），该怎么做呢？这样用户就可以输入自己喜欢的任何口味了。EasyGui 提供了一种输入框（enter box，enterbox）能够做到这一点。可以试试代码清单 6-3 中的程序。

代码清单 6-3　使用输入框得到输入

```
import easygui
flavor = easygui.enterbox("What is your favorite ice cream flavor?")
easygui.msgbox ("You entered " + flavor)
```

运行这个程序时，你会看到：



然后键入你最喜欢的口味，点击 OK，就像前面一样，你键入的内容会显示在消息框中。

这就类似于 raw_input()，同样可以从用户得到文本（一个字符串）。

默认输入

有时用户输入信息时，可能会期望得到某个答案，或者有一个很常见或最可能输入的答案。这称为默认值（default）。这个最常见的答案可以由你为用户自动输入，这样用户就不用再键入了。有了默认值，只有当用户有不同的输入时才有必要键入。

要在一个输入框中放入默认值，可以按照代码清单 6-4 修改你的程序。

代码清单 6-4　如何建立默认参数

```
import easygui
flavor = easygui.enterbox("What is your favorite ice cream flavor?",
                          default = 'Vanilla')        ← 这里是默认值
easygui.msgbox ("You entered " + flavor)
```

现在运行这个程序时，输入框中已经输入了“Vanilla”（香草）。可以把它删掉，再输入你想要的内容，不过如果你最喜欢的口味确实是香草，就不用再键入任何内容，只需点击 OK。

数字呢

如果想在 EasyGui 中输入一个数，完全可以先通过输入框得到一个字符串，然后使用 `int()` 或者 `float()` 由这个字符串创建一个数（就像第 4 章中的做法一样）。

EasyGui 还提供了一种整数框（integer box，`integerbox`），可以用它来输入整数。还可以对所输入的数设置一个下界和上界。

不过，整数框不允许输入浮点数（小数）。要输入小数，必须先通过输入框得到字符串，然后再使用 `float()` 转换这个字符串。

6.5　再看猜数游戏……

第 1 章中，我创建了一个简单的猜数程序。下面再来完成这个程序，不过这一次我们要使用 EasyGui 完成输入和输出。代码清单 6-5 显示了这个程序的代码。

代码清单 6-5　使用 EasyGui 的猜数游戏

```
import random, easygui
secret = random.randint(1, 99)         ← 选一个秘密数
guess = 0
tries = 0
easygui.msgbox("""AHOY!  I'm the Dread Pirate Roberts, and I have a secret!
It is a number from 1 to 99.  I'll give you 6 tries.""")
while guess != secret and tries < 6:                        得到玩家猜的数
    guess = easygui.integerbox("What's yer guess, matey?")   ←
```

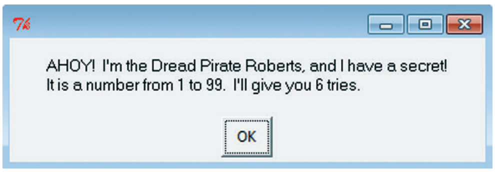
```
    if not guess: break
    if guess < secret:
        easygui.msgbox(str(guess) + " is too low, ye scurvy dog!")
    elif guess > secret:
        easygui.msgbox(str(guess) + " is too high, landlubber!")
    tries = tries + 1
if guess == secret:
    easygui.msgbox("Avast! Ye got it!  Found my secret, ye did!")
else:
    easygui.msgbox("No more guesses!  The number was " + str(secret))
```

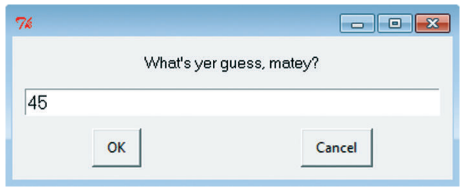
最多允许猜 6 次

用掉一次机会

游戏结束时打印消息

我们还没有全面学习这个程序中各个部分是如何工作的，不过你可以先键入这个程序，试试看。运行程序时你会看到：





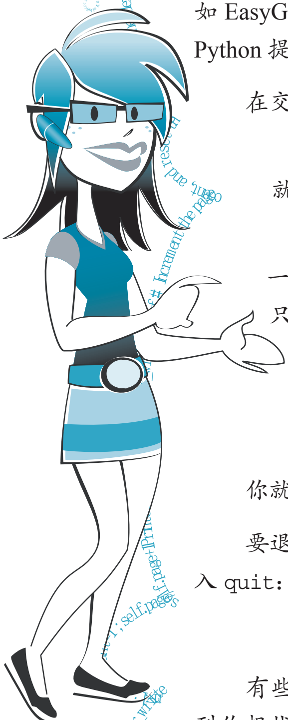
我们将在第 7 章学习 if、else 和 elif 的内容。第 8 章介绍 while，random 会在第 15 章讲到。另外我们还会在第 23 章大量使用 random。

6.6 其他 GUI 组件

EasyGui 还提供了另外一些 GUI 组件，包括允许多重选择（而不是只选择一项）的选择框，还有一些特殊的对话框用来得到文件名等内容。不过，对现在来说，前面介绍的 GUI 组件已经足够了。

利用 EasyGui，我们可以非常容易地生成一些简单的 GUI，而且它隐藏了 GUI 涉及的很多复杂性，使你不用再操心这些问题。后面我们将会讨论建立 GUI 的另一种方法，它可以提供更多的灵活性和控制。

如果你想更多地了解 EasyGui，可以访问 EasyGui 主页 easygui.sourceforge.net。



像（Python）程序员一样思考

如果你想了解有关 Python 使用的更多内容，比如 EasyGui（或任何其他方面），有一个好消息告诉你：Python 提供了一个内置的帮助系统，也许你可以试一试。

在交互模式中，可以在交互提示符后面键入

```
>>>help()
```

就会进入这个帮助系统。现在提示符会变成：

```
help >
```

一旦进入帮助系统，你想要得到哪方面的帮助，只需要键入相应的名字，例如：

```
help> time.sleep
```

或者

```
help> easygui.msgbox
```

你就会得到你想要的一些信息。

要退出帮助系统，重回正常的交互提示符，只需要键入 quit：

```
help> quit
>>>
```

有些帮助读起来很费劲，也很难理解，你往往找不到你想找的东西。不过如果你要找 Python 中某个方面的更多信息，这个帮助系统还是值得试一试。

你学到了什么

在这一章，你学到了以下内容。

- ❒ 如何利用 EasyGui 建立简单的 GUI。
- ❒ 如何使用消息框 msgbox 显示消息。

- 如何使用按钮、选择框和文本输入框（`buttonbox`、`choicebox`、`enterbox`、`integerbox`）得到输入。
- 如何为一个文本框设置默认输入。
- 如何使用 Python 的内置帮助系统。

测试题

1. 如何使用 EasyGui 生成消息框？
2. 如何使用 EasyGui 得到字符串（一些文本）输入？
3. 如何使用 EasyGui 得到整数输入？
4. 如何使用 EasyGui 得到浮点数（小数）输入？
5. 什么是默认值？给出一个可能使用默认值的例子。

动手试一试

1. 试着修改第 5 章中的温度转换程序，这一次要用 GUI 输入和输出而不是 `raw_input()` 和 `print`。

2. 编写一个程序，询问你的姓名，然后是房间号、街道和城市，接下来是省 / 地区 / 州，最后是邮政编码（所有这些都放在 EasyGui 对话框中）。然后这个程序要显示一个寄信格式的完整地址，类似于：

```
John Snead
28 Main Street
Akron, Ohio
12345
```

第 7 章
判断再判断

在前几章中，我们已经看到了程序的一些基本构成模块。现在可以利用输入、处理和输出建立一个程序了。我们甚至还可以通过使用 GUI 让输入和输出更有意思一些。我们可以把输入赋给一个变量，以便以后使用，还可以使用一些数学运算来进行处理。现在来看可以通过哪些方法对程序的工作进行控制。

如果一个程序每次都做同样的事情，这会有些枯燥，而且用处不大。程序要能够决定接下来做什么。我们已经掌握了一些处理技术，下面再来补充另外一些决策（decision-making）技术。

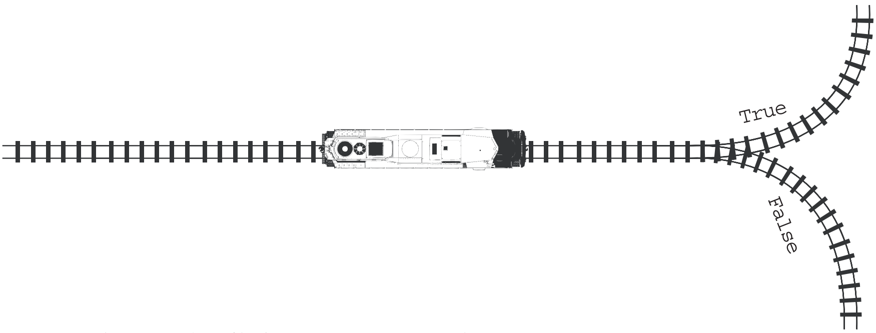
7.1 测试，测试

程序需要能够根据输入做不同的事情。下面给出几个例子：

- 如果 Tim 给出的答案正确，就为他加 1 分；
- 如果 Jane 击中外星人，就发出爆炸声；
- 如果文件没找到，就显示错误消息。

决策时，程序要做出检查（完成一个测试），查看某个条件是否为真。在上面的第一个例子中，这个条件就是“答案正确”。

Python 完成测试的方法很有限，而且每个测试只有两个可能的答案：真（true）或者假（false）。



Python 在测试时可能会问下面这些问题。

- 这两个东西相等吗？
- 其中一个是不是小于另一个？
- 其中一个是不是大于另一个？

不过等一下，刚才说过第一个例子的测试条件是“答案正确”，但是这不属于我们能做的测试，至少不能直接测试。这说明，我们需要用一种 Python 能理解的方式来描述测试。

想要知道 Tim 的答案是否正确时，我们需要知道正确的答案是什么，还要知道 Tim 的答案。可以写成这种形式：



如果 Tim 的答案是正确的，这两个变量就是相等的，所以条件（condition）为真（true）。如果他的答案不正确，这两个变量就不相等，条件则为假（false）。

术语箱

完成测试并根据结果做出判断称为分支（branching）。程序根据测试的结果来决定走哪条路，或者沿哪个分支执行。

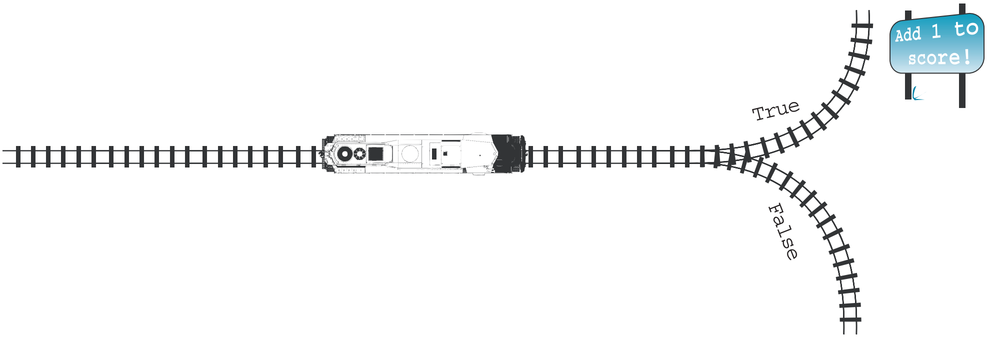
Python 使用关键字 `if` 来测试条件，如下：

```
if timsAnswer == correctAnswer:
    print "You got it right!"
    score = score + 1
print "Thanks for playing."
```

这些代码行构成一个代码“块”，因为相对于上面和下面的代码行已经将它们缩进。

术语箱

代码块（block）是一行或放在一起的多行代码。它们都与程序的某个部分相关（比如一个 `if` 语句）。在 Python 中，通过将块中的代码行缩进来构成代码块。



`if` 行末尾的冒号告诉 Python 下面将是一个指令块。这个块包括从前面的 `if` 行以下直到下一个不缩进的代码行之间的所有缩进代码行。

术语箱

缩进（indenting）是指一个代码行稍稍靠右一点。它不是从最左端开始，而是前面有一些空格，所以会从距左边界几个字符之后开始。

如果条件为真，就会完成之后代码块中的所有工作。在前面的小例子中，第 2 行和第 3 行构成了第 1 行中 `if` 的相应语句块。

现在来讨论缩进和代码块。

7.2　缩进

有些语言中，缩进只是一个风格问题，不论你喜欢还是不喜欢，都可以缩进。不过，在 Python 中，编写代码时缩进是必不可少的一部分。缩进会告诉 Python 代码块从哪里开始，到哪里结束。

Python 中的一些语句（如 if 语句）需要一个代码块来告诉它们要做什么。对于 if 语句，代码块会告诉 Python 如果条件为真时做什么。

将代码块缩进多远并不重要，只要保证整个代码块缩进的程度是一样的。Python 中有一个惯例：总是将代码块缩进 4 个空格。在你的程序中最好也采用这种风格。

术语箱

惯例（convention）就是大多数人都采用的做法。

7.3 是不是有问题



if 语句中真的有两个等号吗（if timsAnswer == correctAnswer）？没错，确实是这样，下面来告诉你这是为什么。

人们通常这么说，“5 加 4 等于 9”，另外会这么问“5 加 4 等于 9 吗？”。前一个是陈述句（statement）；另一个是疑问句（question）。

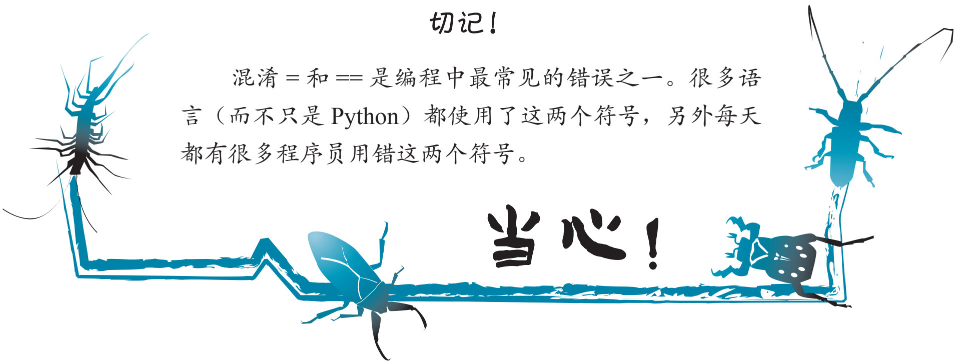
在 Python 中，也同样有陈述句（或语句）以及疑问句（或问题）。语句可能将值赋给一个变量。问题可能查看变量是否等于某个值。前者是在做某种设置（赋值或设置为相等），后者在做某种检查或测试（是否相等，对还是错），所以 Python 使用了两种不同的符号。

我们已经看到，等号（=）用来设置变量或赋值。下面再给出几个例子：

```
correctAnswer = 5 + 3
temperature = 35
name = "Bill"
```

要测试两个东西是否相等，Python 使用了一个双等号（==），如下：

```
if myAnswer == correctAnswer:
if temperature == 40:
if name == "Fred":
```



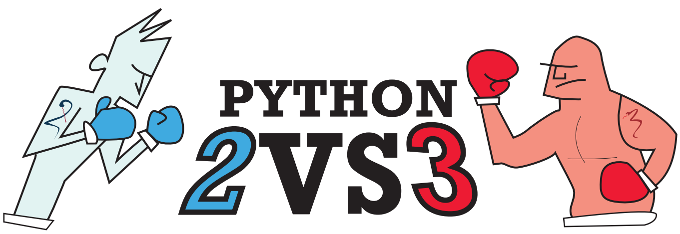
切记！

混淆 = 和 == 是编程中最常见的错误之一。很多语言（而不只是 Python）都使用了这两个符号，另外每天都有很多程序员用错这两个符号。

测试或检查也称为比较。双等号称为一个比较操作符。应该记得，我们在第 3 章讨论过操作符。操作符就是会对两边的值进行操作的一个特殊符号。在这里，操作就是测试两个值是否相等。

7.4　其他类型的测试

很幸运，其他比较操作符更容易记：小于（<）、大于（>）和不等于（!=）。（还可以使用 <> 表示不等于，不过大多数人都用 !=。）还可以把 > 或 < 与 = 结合起来表示大于或等于（>=）以及小于或等于（<=）。数学课上你可能已经见过这样一些符号。



不等于

在 Python 3 中，不再支持 <> 形式的不等于，只能使用 != 来表示不等于。

还可以把两个大于和小于操作符“串”在一起完成一个范围测试，比如：

```
if 8 < age < 12:
```

这会检查变量 age 的值是否介于（但不包含）8 和 12 之间。如果 age 等于 9、10 或 11（或者 8.1 或 11.6 等），这就会是 true。如果希望包含年龄为 8 和 12 的情况，可以这样做：

```
if 8 <= age <= 12:
```

术语箱

比较操作符（comparison operator）也称为关系操作符（relational operator），因为它们要测试两边值的关系（relation）：相等还是不相等，大于还是小于。比较也称为条件测试（conditional test）或逻辑测试（logical test）。在编程中，逻辑（logical）就是指某个结论的答案是真还是假。

代码清单 7-1 显示了一个使用比较的示例程序。先在 IDLE 编辑器中创建一个新文件，键入这个程序并保存，把它命名为 compare.py。然后运行这个程序。试着用不同的数运行多次。可以试试不同的情况，比如第一个数较大、第一个数较小，以及两个数相等，看看会得到什么结果。

代码清单 7-1　使用比较操作符

```
num1 = float(raw_input("Enter the first number: "))
num2 = float(raw_input("Enter the second number: "))
if num1 < num2:
    print num1, "is less than", num2
if num1 > num2:
    print num1, "is greater than", num2
if num1 == num2:
    print num1, "is equal to", num2
if num1 != num2:
    print num1, "is not equal to", num2
```

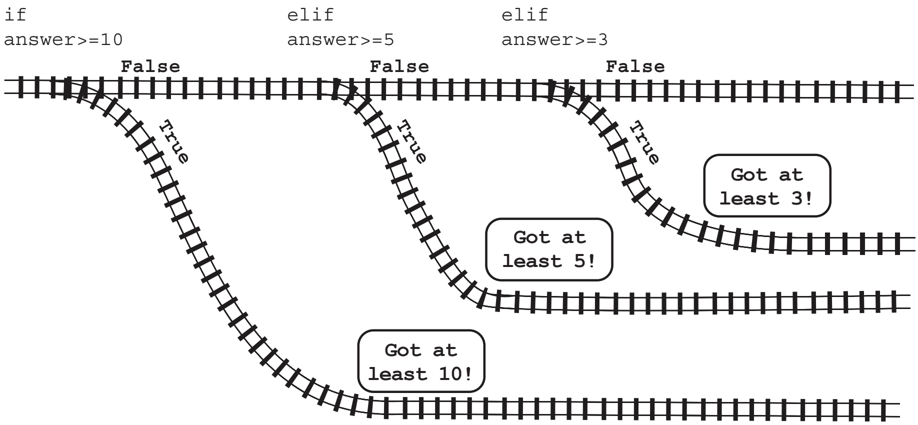
记住这是一个双等号

7.5　如果测试为假会怎么样

我们已经看到了，如果测试的结果为真，Python 会做些什么。不过，如果测试为假，Python 又会做一些什么呢？在 Python 中，有以下 3 种可能。

- 做另一个测试。如果第一个测试结果为假，可以利用关键字 `elif`（这是 else if 的简写）让 Python 再做另一个测试，例如：

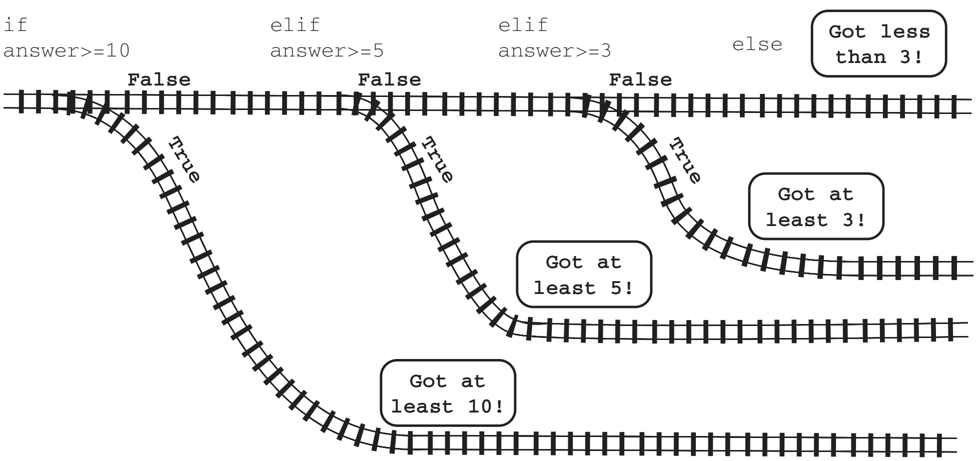
```
if answer >= 10:
    print "You got at least 10!"
elif answer >= 5:
    print "You got at least 5!"
elif answer >= 3:
    print "You got at least 3!"
```



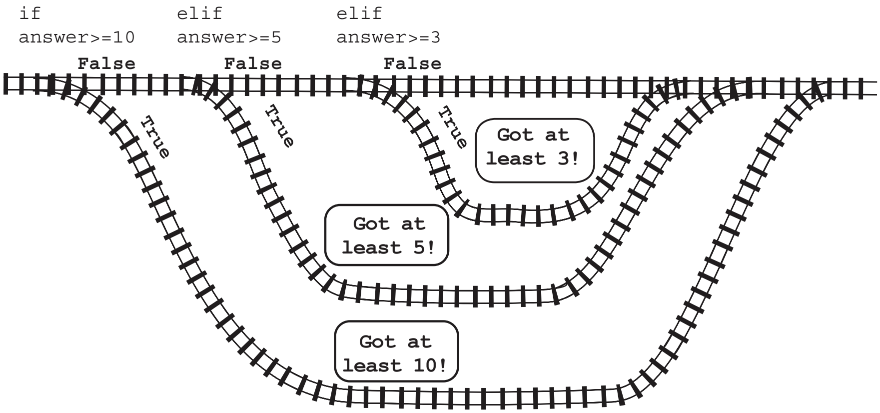
在 if 后面，elif 语句你想要有多少就可以有多少。

- 如果所有其他测试结果都是假，做其他工作。这要利用 else 关键字完成。它总是在最后出现，也就是完成 if 和所有 elif 语句之后。

```
if answer >= 10:
    print "You got at least 10!"
elif answer >= 5:
    print "You got at least 5!"
elif answer >= 3:
    print "You got at least 3!"
else:
    print "You got less than 3."
```



- 继续。如果 if 块后面没有放任何其他东西，程序会继续执行下一行代码（如果有的话），或者会结束（如果再没有更多代码）。



试着用上面的代码建立一个程序，在最开始增加一行代码输入一个数：

```
answer = float(raw_input ("Enter a number from 1 to 15"))
```

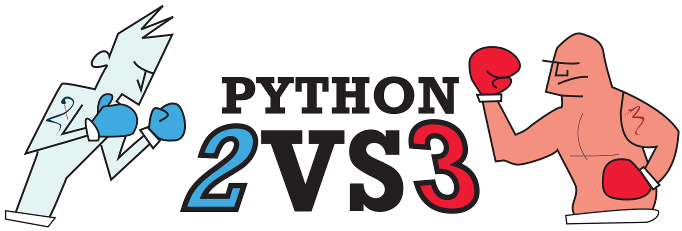
记住要保存这个文件（这一次由你来选择文件名），再运行这个程序。用不同的输入多试几次，看看会得到什么结果。

7.6　测试多个条件

如果想要测试好几件事情该怎么办？假设你要为 8 岁以上的人创建一个游戏，另外你希望玩家至少上三年级。这就要满足两个条件。下面是测试这两个条件的一种方法：

```
age = float(raw_input("Enter your age: "))
grade = int(raw_input("Enter your grade: "))
if age >= 8:
    if grade >= 3:
        print "You can play this game."
else:
    print "Sorry, you can't play the game."
```

注意第一个 print 行缩进 8 个空格，而不只是 4 个空格。这是因为每个 if 都需要自己的代码块，所以都要缩进 4 个空格。



提示

记住，如果你使用 Python 3 的话，需要将 raw_input() 替换为 input()，调用 print 时也需要加上括号，像这样：

```
print("You can play this game.")
```

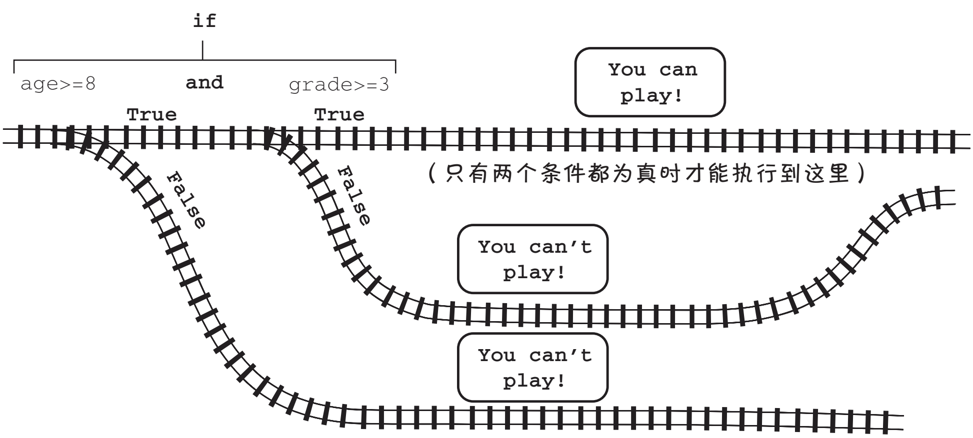
7.7　使用 and

上面最后这个例子可以达到目的。不过还有一种更简便的方法来做同样的事情。可以像下面这样结合两个条件：

```
age = float(raw_input("Enter your age: "))
grade = int(raw_input("Enter your grade: "))
if age >= 8 and grade >= 3:
    print "You can play this game."
else:
    print "Sorry, you can't play the game."
```

用“and”结合多个条件

我们使用 and 关键字来结合这两个条件。and 表示两个条件都必须为真才能执行下面的代码块。



可以用 and 把两个以上的条件放在一起：

```
age = float(raw_input("Enter your age: "))
grade = int(raw_input("Enter your grade: "))
color = raw_input("Enter your favorite color: ")
if age >= 8 and grade >= 3 and color == "green":
    print "You are allowed to play this game."
else:
    print "Sorry, you can't play the game."
```

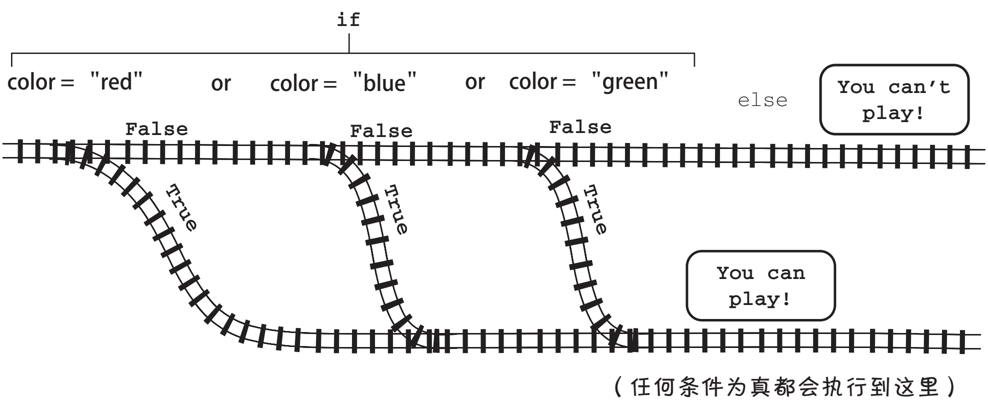
如果有两个以上的条件，所有条件都必须为真，这个 if 语句才能为真。

还有其他方法来结合多个条件。

7.8 使用 or

or 关键字也是用来把多个条件放在一起。如果使用 or，只要任意一个条件为真，就会执行代码块。

```
color = raw_input("Enter your favorite color: ")
if color == "red" or color == "blue" or color == "green":
    print "You are allowed to play this game."
else:
    print "Sorry, you can't play the game."
```



7.9 使用 not

还可以用 not 把比较倒过来，表示相反的逻辑。

```
color = raw_input("Enter your favorite color: ")
if color == "red" or color == "blue" or color == "green":
    print "You are allowed to play this game."
else:
    print "Sorry, you can't play the game."
```

这行代码

```
if not (age < 8):
```

与这行代码：

```
if age >= 8:
```

两者的含义是一样的。在这两种情况下，如果年龄是 8 岁或者超过 8 岁就会执行代码块，如果年龄小于 8 就不会执行。

第 4 章中，我们见过诸如 +、-、* 和 / 等数学运算符。在本章中，我们了解了比较操作符 <、>、== 等。and、or 和 not 关键字也是操作符。它们被称为逻辑操作符（logical operator）。这些操作符用来修改比较，可以结合两个或多个比较（and 和 or），或者取反（not）。

表 7-1 列出了目前为止我们讨论过的所有操作符。

表 7-1 数学和比较操作符列表

操作符	名　字	作　用
数学运算符		
=	赋值	将一个值赋至一个名（变量）
+	加	两个数相加。这个操作符也可以用来连接字符串
-	减	两个数相减
+=	自增	将一个数增 1
-=	自减	将一个数减 1
*	乘	两个数相乘
/	除	两个数相除。如果两个数都是整数，结果只是整数商而没有余数
%	取余	得到两个数整除的余数
**	求幂	将一个数自乘得到幂。这个数以及幂可以是整数或浮点数
比较操作符		
==	相等	检查两个东西是否相等
<	小于	检查第一个数是否小于第二个数
>	大于	检查第一个数是否大于第二个数
<=	小于或等于	检查第一个数是否小于或等于第二个数
>=	大于或等于	检查第一个数是否大于或等于第二个数
!=	不等于	检查两个东西是否不相等（这两个操作符都可以使用）
<>		

你可能想在这一页上夹个书签，这样下次就能很容易地查阅这个表了。

你学到了什么

在这一章，你学到了以下内容。

- 比较测试和关系操作符。
- 缩进和代码块。
- 使用 and 和 or 结合测试。
- 使用 not 来进行反向测试。

测试题

1. 运行这个程序会得到什么输出：

```
my_number = 7
if my_number < 20:
    print 'Under 20'
else:
    print '20 or over'
```

2. 基于第一个问题中的程序，如果把 my_number 改为 25，输出会是什么？
3. 要检查一个数是否大于 30 但小于或等于 40，要用哪种 if 语句？
4. 要检查用户输入的字母“Q”是大写还是小写，要使用哪种 if 语句？

动手试一试

1. 一家商场在降价促销。如果购买金额低于或等于 10 元，会给 10% 的折扣，如果购买金额大于 10 元，会给 20% 的折扣。编写一个程序，询问购买价格，再显示折扣（10% 或 20%）和最终价格。
2. 一个足球队在寻找年龄在 10 到 12 岁之间的小女孩加入。编写一个程序，询问用户的年龄和性别（m 表示男性，f 表示女性）。显示一条消息指出这个人是否可以加入球队。

   额外提示：要合理地建立程序，如果用户不是女孩就不必询问年龄。
3. 你在长途旅行，刚到一个加油站，距下一个加油站还有 200 km。编写一个程序确定是不是需要在这里加油，还是可以等到下一个加油站再加油。

   这个程序应当问下面几个问题。
   - 你的油箱有多大（单位是升）？
   - 油箱有多满（按百分比，例如，半满就是 50%）？
   - 你的汽车每升油可以走多远（km）？

   输出应该像这样：

```
Size of tank:  60
percent full:  40
km per liter:  10
You can go another 240 km
The next gas station is 200 km away
You can wait for the next station.
```

或

```
Size of tank:  60
percent full:  30
km per liter:  8
You can go another 144 km
The next gas station is 200 km away
Get gas now!
```

额外提示：程序中包含一个 5 升的缓冲区，以防油表不准。

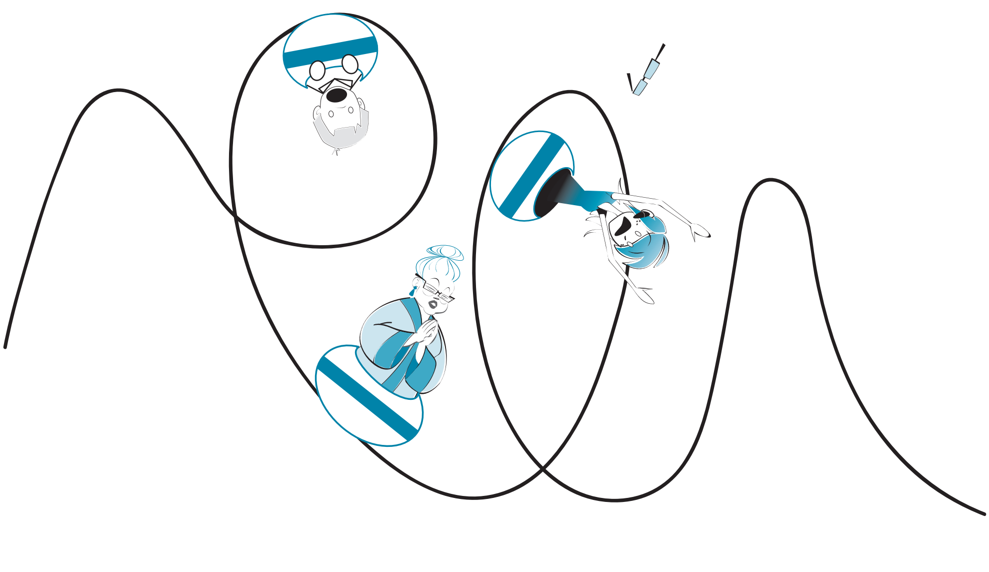
4. 建立一个程序，用户必须输入密码才能使用这个程序。你当然知道密码（因为它会写在你的代码中）。不过，你的朋友要得到这个密码就必须问你或者直接猜，也可以学习足够的 Python 知识查看代码来找出密码！
对程序没什么要求，可以是你已经编写的程序，也可以是一个非常简单的程序，只在用户输入正确的口令时显示一条“You’re in!”之类的消息。

第 8 章
转　圈　圈

对大多数人来说，反复地做同样的事情很烦人，既然如此，为什么不让计算机来为我们做这些事情呢？计算机从来不会觉得烦，所以它们非常擅长去完成重复的任务。在这一章中我们就来看如何让计算机做重复的事情。

计算机程序通常会周而复始地重复同样的步骤，这称为循环（looping）。主要有两种类型的循环：

- 重复一定次数的循环，称为计数循环（counting loop）；
- 重复直至发生某种情况时结束的循环，称为条件循环（conditional loop），因为只要条件为真，这种循环会一直持续下去。



8.1　计数循环

第一种循环称为计数循环。我们还听过有人把它叫做 for 循环，因为很多语言（包括 Python）在程序中都使用 for 关键字来创建这种类型的循环。

下面就来尝试使用计数循环的程序。在 IDLE 中使用 File（文件）➤ New（新建）命令打开一个新的文本编辑器窗口（就像写第一个程序时一样）。然后键入代码清单 8-1 中的程序。

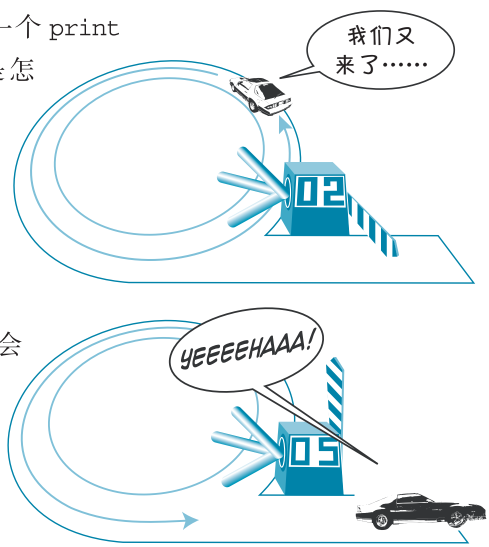
代码清单 8-1　一个非常简单的 for 循环

```
for looper in [1, 2, 3, 4, 5]:
    print "hello"
```

把它保存为 Loop1.py，运行这个程序（可以使用 Run（运行）➤ Run Module（运行模块）菜单，也可以用快捷键 F5）。

你会看到这样的结果：

```
>>> ================ RESTART ================
>>>
hello
hello
hello
hello
hello
```



嘿，是不是有重复？虽然这里只有一个 print 语句，但程序显示了 5 次“hello”。这是怎么做到的？第一行（for looper in [1, 2, 3, 4, 5]:）翻译成我们的语言就表示。

(1) 变量 looper 的值从 1 开始（所以 looper = 1)。

(2) 对应列表中的每一个值，这个循环会把下一个指令块中的所有工作完成一次。(列表就是中括号中的那些数)。

(3) 每次执行循环时，变量 looper 会赋为这个列表中的下一个值。

第二行（print "hello"）就是 Python 每次循环时要执行的代码块。for 循环需要一个代码块来告诉程序每次循环时做什么。这个代码块（缩进的代码部分）称为循环体（body of the loop）。（还记得吧？上一章我们讨论过缩进和代码块。）

术语箱

每次循环称为一次迭代（iteration）。

下面来试试别的。这一次不再是每次都打印相同的东西，下面让它每次循环时打印不同的东西。代码清单 8-2 就会做这个工作。

代码清单 8-2　每次 for 循环做不同的事情

```
for looper in [1, 2, 3, 4, 5]:
    print looper
```

把这个程序保存为 Loop2.py，并运行。

结果应该类似于：

```
>>> ================ RESTART ================
>>>
1
2
3
4
5
```

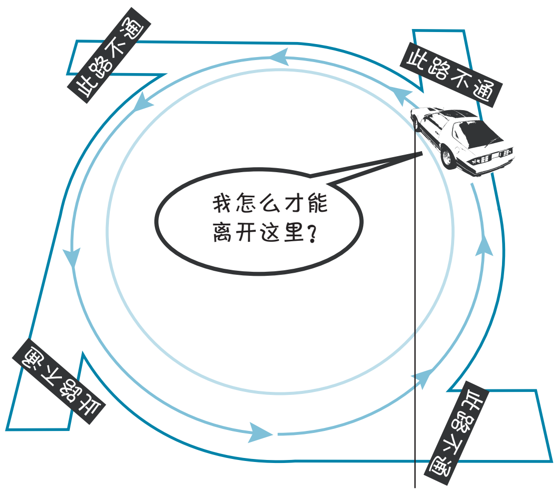
这一次不再打印 5 次“hello”了，它会打印变量 looper 的值。每次循环时，looper 会取列表中的下一个值。



失控的循环

卡特，我也遇到过同样的问题！每一个程序员都曾经遭遇过失控的循环（也叫做无限循环）。任何时刻（甚至在失控循环中）要停止一个 Python 程序，只需要按下 CTRL-C，即按下 CTRL 键的同时再按下 C 键。以后你会发现这非常方便！游戏和图形程序通常都在一个循环中运行。这些程序需要不断从鼠标、键盘或游戏控制器得到输入，然后处理这个输入，并更新屏幕。开始写这种程序时，我们会大量使用循环。所以你的某个程序

很有可能会在某一点陷入循环，所以你要知道如何让它脱身！



中括号做什么用

你可能已经注意到，循环值的列表包围在中括号里。Python 就是利用中括号以及数之间的逗号来建立列表（list）。稍后就会学习列表（准确地讲，是在第 12 章）。不过对现在来说，只要知道列表就是一种“容器”，可以用来存放一堆东西。在这里，这些东西就是数，也就是每次循环迭代时 looper 所取的值。

8.2　使用计数循环

现在利用循环做点有意义的事情。下面打印一个乘法表。这里只对前面的程序做一个小小的修改。这个新版本的程序见代码清单 8-3。

代码清单 8-3　打印 8 的乘法表

```
for looper in [1, 2, 3, 4, 5]:
    print looper, "times 8 =", looper * 8
```

把这个程序保存为 Loop3.py，然后运行。你会看到这样的结果：

```
>>> =================== RESTART ===================
>>>
1 times 8 = 8
2 times 8 = 16
3 times 8 = 24
4 times 8 = 32
5 times 8 = 40
```

现在我们终于见识了循环的威力。如果没有循环，要得到同样的结果必须编写这样一个程序：

```
print "1 times 8 =", 1 * 8
print "2 times 8 =", 2 * 8
print "3 times 8 =", 3 * 8
print "4 times 8 =", 4 * 8
print "5 times 8 =", 5 * 8
```

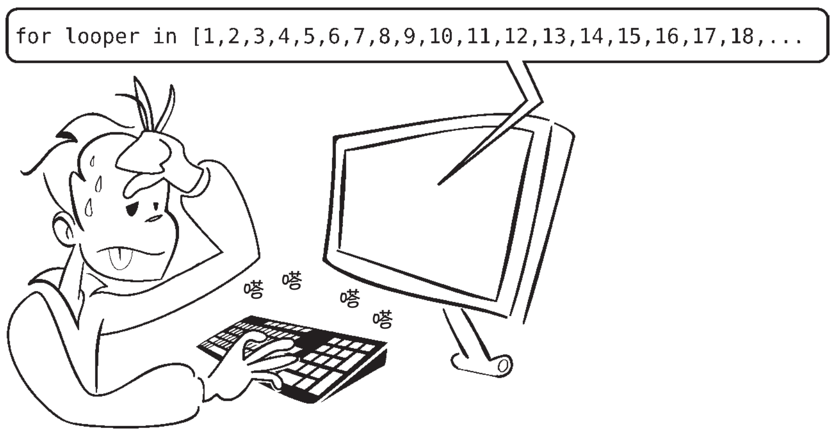
要建立一个更长的乘法表（比如说，从 1 到 10 或者到 20），这个程序可能会更长，不过我们的循环程序几乎不变（只不过列表中会有更多的数）。循环使问题简单多了！

8.3　一条捷径——range()

在上面的例子中，我们只循环了 5 次：

```
for looper in [1, 2, 3, 4, 5]:
```

如果希望循环运行 100 次或者 1000 次呢？这就得键入很多很多的数！



很幸运，这里有一条捷径。利用 range() 函数，你可以只输入起始值和结束值，它就会为你创建这二者之间的所有值。range() 会创建一个列表，其中包含某个范围内的数。

代码清单 8-4 仍然使用我们在乘法表中用到的例子，不过这里使用了 range() 函数。

代码清单 8-4　使用 range() 的循环

```
for looper in range (1, 5):
    print looper, "times 8 =", looper * 8
```

把这个程序保存为 Loop4.py 并运行（可以使用 Run（运行）➤ Run Module（运行模块）菜单，或者按下快捷键 F5）。你会看到这样的结果：

```
>>> ================= RESTART =================
>>>
1 times 8 = 8
2 times 8 = 16
3 times 8 = 24
4 times 8 = 32
```

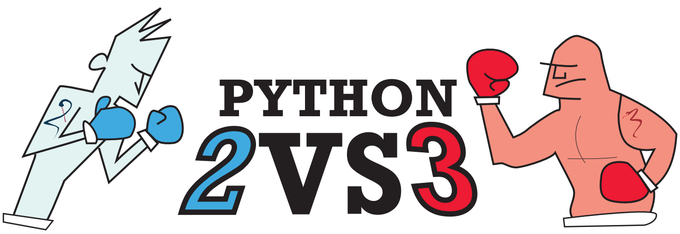
基本上与第一个结果完全相同……不过少了最后一次循环！为什么呢？

答案在于，range (1, 5) 给出的列表是 [1, 2, 3, 4]。你可以在交互模式中试试看：

```
>>> print range(1, 5)
[1, 2, 3, 4]
```

为什么没有 5 呢？

嗯，这正是 range() 函数的做法。它会提供一个数字列表，从给定的第一个数开始，在给定的最后一个数之前结束。必须考虑到这一点，调整范围来得到想要的循环次数。



Range

如果你在 Python 3 中这样做，结果会有点不一样：

```
>>> print(range(1, 5))
range(1,5)
```

这是因为在 Python 3 中 range() 函数不会提供一个数字列表，而是会提供一个“可迭代”（iterable）的东西，你可以使用循环来遍历它。（前文的“术语箱”中提到过的。）

如果在 for 循环中使用 range()，则其工作方式是完全一样的，只是内部机制略有不同而已。

代码清单 8-5 给出了修改后的程序，它会给出 8 的乘法表（从 1 到 10）。

代码清单 8-5 使用 range() 打印 8 的乘法表（从 1 到 10）

```
for looper in range(1, 11):
    print looper, "times 8 =", looper * 8
```

运行这个程序时会得到下面的结果：

```
>>> ================== RESTART ==================
>>>
1 times 8 = 8
2 times 8 = 16
3 times 8 = 24
4 times 8 = 32
5 times 8 = 40
6 times 8 = 48
7 times 8 = 56
8 times 8 = 64
9 times 8 = 72
10 times 8 = 80
```

在代码清单 8-5 的程序中，`range(1, 11)` 给出从 1 到 10 的一个数字列表，对于列表中的每一个数会完成一次循环迭代。每次循环时，变量 `looper` 就取列表中的下一个值。

顺便说一句，我们把循环变量叫做 `looper`，不过也可以取你喜欢的任何名字。

8.4 风格问题——循环变量名

循环变量与其他变量一样。它没有任何特殊之处，只是对应一个值的名字而已。将这个变量用作循环计数器也是可以的。

之前我们说过，要使用能够描述变量用途的变量名。正是这个原因，我们在前一个例子中选择了 `looper` 这个名字。不过，有时可以有些例外，循环变量就属于这种例外。这是因为，编程中有一个惯例（应该记得，惯例就是表示通用的做法），通常使用字母 `i`、`j`、`k` 等作为循环变量。

从前的美好时光



为什么用 i、j 和 k 循环？

这是因为早先的程序员一直用程序来计算数学问题，而数学中 a、b、c 和 x、y、z 已经有其他用途。另外，在当时一种流行的编程语言中，变量 i、j 和 k 总是整数，不能把它们创建为任何其他类型。由于循环计数器总是整数，所以程序员总是选择 i、j 和 k 来作为循环计数器，这也成为了一种通用的做法。

由于很多人都使用 i、j、k 作为循环变量，程序员在程序中也习惯了这种做法。当然也可以用其他名字作为循环变量（就像前面的例子中一样），不过，除了作为循环变量，i、j、k 不应当有其他用途。

如果采用这个惯例，程序就会像这样：

```
for i in range (1, 5):
    print i, "times 8 =", i * 8
```

它的用法完全相同。（你可以试试看！）

为循环变量选择什么名字属于风格问题。风格（style）就是你的程序看上去怎么样，而与程序能不能正常工作无关。不过，如果与其他程序员采用相同的风格，你的程序就会更易读、更易于理解，也更易于调试。同时，你也会更加习惯这种风格，能够更轻松地读懂其他人的程序。

range() 简写

不一定非得为 range() 提供两个数（像在代码清单 8-5 中那样），可以只提供一个数：

```
for i in range (5):
```

这与写作：

```
for i in range (0, 5):
```

完全相同，同样会提供以下数字列表：[0, 1, 2, 3, 4]。

实际上，大多数程序员都从 0 开始循环而不是从 1 开始。如果使用 range(5)，会得到循环的 5 次迭代，这很容易记住。只是需要知道，第一次循环时 i 将等于 0 而不是 1，而最后一次循环时，它将等于 4 而不是 5。

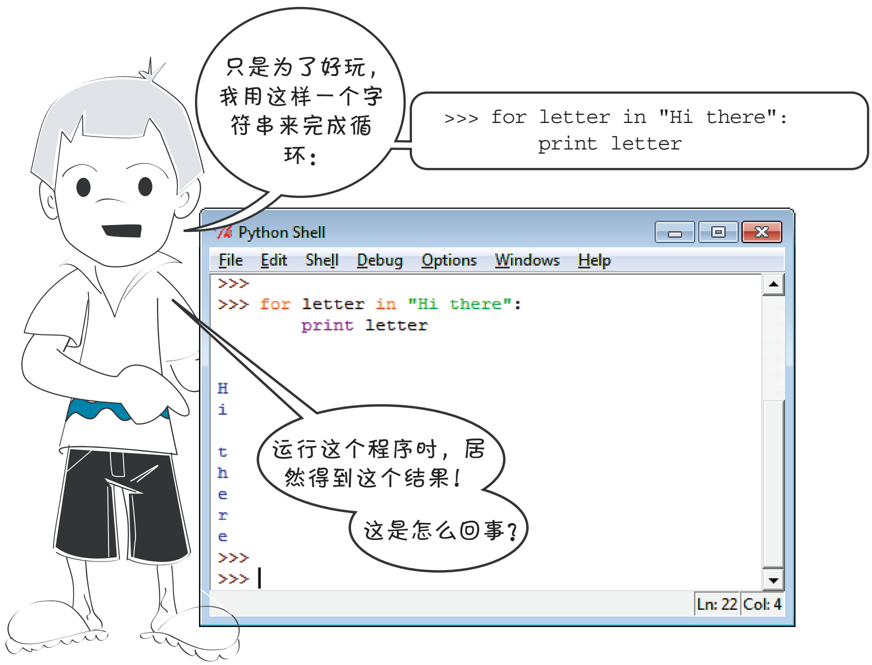
从前的美好时光

为什么大多数程序员从 0 而不是 1 开始循环呢？



是这样的，从前，有些人坚持从 1 开始，有些人则坚持从 0 开始。他们对于哪一种做法更好有过激烈的争论。最终，坚持从 0 开始的人胜利了。

所以就出现了现在的情况，如今大多数人都从 0 开始循环，不过你可以根据自己的喜好选择任何一种做法。只是要记住，需要调整上界来得到正确的迭代次数。



嗯，卡特，你已经发现字符串的一些规律了。字符串就像一个字符列表，我们已经学过：计数循环使用列表来完成迭代。这说明，也可以利用一个字符串来循环。字符串中的每个字符对应循环中的一次迭代。所以，如果打印循环变量（在这个例子中卡特把他的循环变量取名为 `letter`），就会打印出这个字符串中的所有字母，一次打印一个字母。因为每个 `print` 语句都会换行，所以每个字母分别打印在单独的一行上。

你可以像卡特一样，多做一些尝试，这是一种很好的学习方法！

8.5 按步长计数

到目前为止，我们的计数循环都是每次迭代时计数增 1。如果希望循环按步长为 2 来计数该怎么做？或者步长为 5 呢？或者 10 呢？还有，如果想反向计数，又该怎么做呢？

`range()` 函数可以有一个额外的参数，利用这个参数可以把步长从默认的 1 改为不同的值。

术语箱

参数（argument）就是使用类似 range() 的函数时放在括号里的值。我们说，向函数传入了参数。有时也用形参（parameter）这个词，如传递形参。我们将在第 13 章了解更多关于函数、参数和形参的内容。

我们想在交互模式中尝试几个循环。键入第一行时，由于末尾有冒号，IDLE 会自动为你缩进下一行，因为它知道 for 循环后面需要有一个代码块。完成这个代码块后，按两次回车键。试试看：

```
>>> for i in range(1, 10, 2):
        print i

1
3
5
7
9
```

这里向 range() 函数增加了第 3 个参数 2。现在循环按步长 2 计数。再来试一个：

```
>>> for i in range(5, 26, 5):
        print i

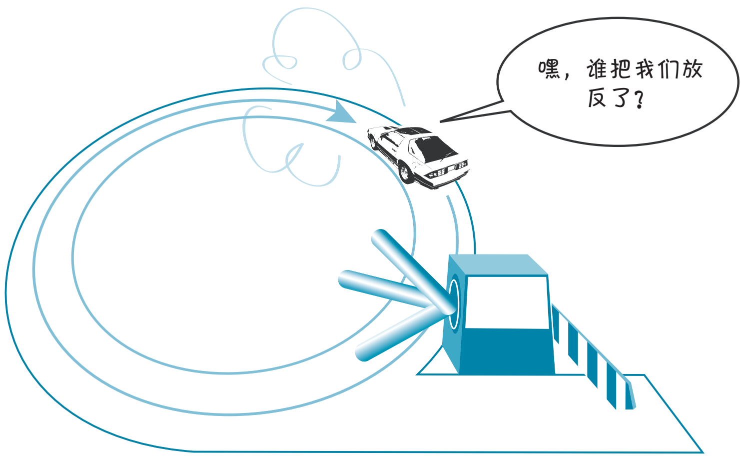
5
10
15
20
25
```

这是按步长 5 来循环的。反向计数呢？

```
>>> for i in range(10, 1, -1):
        print i

10
9
8
7
6
5
4
3
2
```

range() 函数中的第 3 个参数是负数时，循环会向下计数，而不是向上计数。应该记得，循环会从一个数开始，向上（或向下）直到（但不包括）第二个数，所以在最后一个例子中，我们只向下计数到 2，而不是 1。



可以利用这一点来建立一个倒计时的定时器程序。只需要再增加两行代码。在 IDLE 中打开一个新的编辑器窗口，键入代码清单 8-6 中的程序。试着运行这个程序。

代码清单 8-6　准备好了吗？

```
import time
for i in range (10, 0, -1):        ← 反向计数
    print i
    time.sleep(1)                   ← 等待 1 秒
print "BLAST OFF!"
```

先不用担心这个程序里还没有讲到的内容，比如说 import、time 和 sleep。所有这些内容都会在后面的章节中讲清楚。你只需要试着运行代码清单 8-6 中的程序，看看它是怎么工作的。这里的关键是 range(10,0,-1) 部分，它会让循环从 10 反向计数到 1。

8.6　没有数字的计数

在所有前面的例子中，循环变量都是一个数。按编程术语来讲，可以这么说：循环迭代处理一个数字列表。但是列表不一定非得是数字列表。从卡特的试验我们看到，它也可以是字符列表（一个字符串），还可以是一个字符串列表，或者是其他列表。

要了解它如何工作，最好的办法就是举个例子来说明。试着运行代码清单 8-7 中的程序，看看会发生什么。

代码清单 8-7　谁最酷？

```
for cool_guy in ["Spongebob", "Spiderman", "Justin Timberlake", "My Dad"]:
    print cool_guy, "is the coolest guy ever!"
```

现在，我们不再是循环处理一个数字列表，这里会循环处理一个字符串列表。而且不再将 i 作为循环变量，我使用的是 cool_guy。每次循环时，循环变量 cool_guy 会取列表中一个不同的值。这仍然是一种计数循环，因为尽管列表不是数字列表，Python 也要统计列表中有多少项来确定循环多少次。（这一次我没有显示输出，你可以自己运行程序来看看结果。）

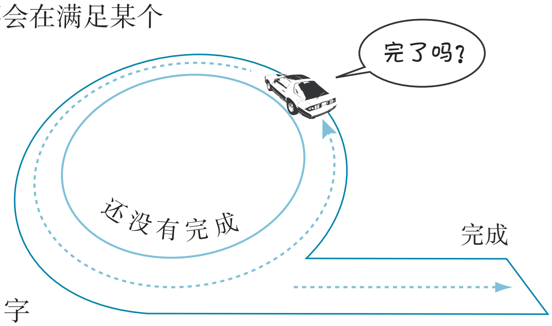
不过，如果我们无法提前知道需要多少次迭代呢？如果没有可用的值列表呢？别着急，接下来就会讲到！

8.7　关于这个问题……

我们刚才学习了第一种的循环，也就是 for 循环或计数循环。第二种循环称为 while 循环或条件循环。

如果你能提前知道希望循环运行多少次，那么 for 循环很合适。不过，有时你可能希望循环一直运行，直到发生某种情况时才结束，而且你不知道发生这种情况之前会有多少次迭代。这就可以使用 while 循环来实现。

上一章中，我们了解了条件和测试，还学习了 if 语句。while 循环并不统计运行多少次循环，它会使用一个测试来确定什么时候停止循环。while 循环也称为条件循环（conditional loop）。条件循环会在满足某个条件时一直保持循环。



基本说来，while 循环会一直问“完了吗？……完了吗？……完了吗？……”，直到完成。它会在条件不再为真时完成。

while 循环使用 Python 关键字 while。代码清单 8-8 给出了一个例子。你可以键入这个程序，试着运行，看看它是如何工作的。（要记住，一定要先保存再运行。）

代码清单 8-8 条件或 while 循环

```
print "Type 3 to continue, anything else to quit."
someInput = raw_input()
while someInput == '3':
    print "Thank you for the 3.  Very kind of you."
    print "Type 3 to continue, anything else to
  quit."
    someInput = raw_input()
print "That's not 3, so I'm quitting now."
```

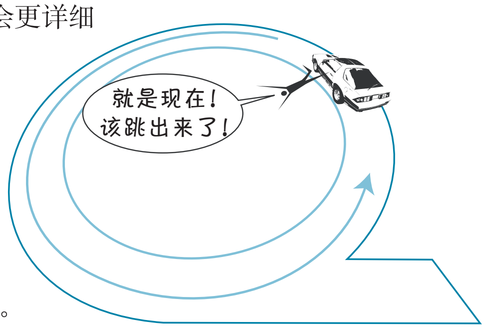
只要 someInput ='3' 就一直循环

循环体

这个程序不断向用户请求输入。当输入等于 3 时，条件为 true，循环继续运行。正是这个原因，这种条件循环也称为 while 循环，它使用了 Python 的 while 关键字。输入不等于 3 时，条件为 false，循环停止。

8.8 跳出循环——break 和 continue

有时可能希望在中间离开循环，也就是 for 循环结束计数之前，或者 while 循环找到结束条件之前。有两种方法来做到：可以用 continue 直接跳到循环的下一次迭代，或者用 break 完全中止循环。下面会更详细地说明。



提前跳转——continue

如果希望停止执行循环的当前迭代，提前跳到下一次迭代，你需要的就是一条 continue 语句。要说明这一点，最好的办法就是看一个例子，请看代码清单 8-9。

代码清单 8-9 循环中使用 continue

```
for i in range (1, 6):
    print
    print 'i =', i,
    print 'Hello, how',
    if i == 3:
        continue
    print 'are you today?'
```

运行这个程序时，输出如下：

```
>>> ================== RESTART ==================
>>>

i = 1 Hello, how are you today?
```

```
i = 2 Hello, how are you today?

i = 3 Hello, how
i = 4 Hello, how are you today?

i = 5 Hello, how are you today?
```

注意，第 3 次循环时（i == 3），循环体没有完成，它提前跳到了下一次迭代（i == 4）。这就是 continue 语句在起作用。在 while 循环中，continue 的作用也是一样的。

跳出——break

如果我们想完全跳出循环——不再完成计数，或者放弃等待结束条件，该怎么做呢？这个工作由 break 语句完成。

下面只改变代码清单 8-9 中的第 6 行，把 continue 换成 break，再运行这个程序看看会发生什么。

```
>>> ================== RESTART ==================
>>>

i = 1 Hello how are you today?

i = 2 Hello how are you today?

i = 3 Hello how
```

这一次，循环不只是跳过第 3 次迭代的其余部分，它会完全停止循环。这正是 break 的作用。在 while 循环中，break 的作用也一样。

要指出的是，有些人认为使用 break 和 continue 并不好。就我个人来讲，我不认为这样不好，不过我自己确实很少使用这两个语句。我想还是应该告诉你一些关于 break 和 continue 的内容，没准以后你会用到。

你学到了什么

在这一章，你学到了以下内容。

- for 循环（也称为计数循环）。
- range() 函数——计数循环的一个捷径。
- range() 的不同步长大小。
- while 循环（也称为条件循环）。

- ❒ 用 continue 跳到下一次迭代。
- ❒ 用 break 跳出循环。

测试题

1. 下面的循环会运行多少次？

```
for i in range (1, 6):
    print 'Hi, Warren'
```

2. 下面的循环会运行多少次？每次循环时 i 的值是什么？

```
for i in range (1, 6, 2):
    print 'Hi, Warren'
```

3. range(1, 8) 会给出一个怎样的数字列表？
4. range(8) 会给出一个怎样的数字列表？
5. range(2, 9, 2) 会给出一个怎样的数字列表？
6. range(10, 0, -2) 会给出一个怎样的数字列表？
7. 使用哪个关键字停止循环的当前迭代，提前跳到下一次迭代？
8. while 循环什么时候结束？

动手试一试

1. 编写一个程序，显示一个乘法表。开始时要询问用户显示哪个数的乘法表。输出应该如下所示：

```
Which multiplication table would you like?
5
Here's your table:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

2. 完成第 1 题的程序时你可能使用了 for 循环。大多数人都会这么做。不过，可以再做个练习，试着用 while 循环完成同样的工作。或者如果你在第 1 题中使用了 while 循环，现在可以试着用 for 循环来完成。

3. 向乘法表程序中再加点东西。询问用户想要的乘法表之后，再问问用户希望最大乘到几。输出应当如下所示：

```
Which multiplication table would you like?
7
How high do you want to go?
12
Here's your table:
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
7 x 11 = 77
7 x 12 = 84
```

可以用 `for` 循环或者 `while` 循环的版本来完成，或者两种做法都试试看。

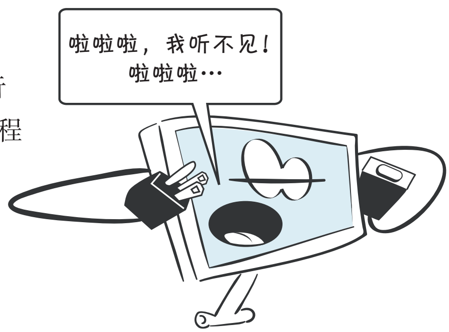
第 9 章

全都为了你——注释

到现在为止，我们在程序（以及交互模式）中键入的所有一切都是交给计算机的指令。不过，还可以在程序中为你自己加入一些说明，描述这个程序做什么，怎么做，这是一个很好的想法。这样能够帮助你（或者其他人）以后查看程序，了解原先你想做什么。

在计算机程序中，这些说明就称为注释（comment）。

9.1 增加注释



注释是给你看的，而不是让计算机执行的。注释是程序文档的一部分，计算机运行程序时会忽略这些注释。

Python 中向程序增加注释有两种方法。

术语箱

文档（documentation）就是关于一个程序的信息，描述了程序并说明它是如何工作的。注释是程序文档的一部分，不过在代码本身以外，文档还包括其他部分，文档描述以下内容：

- ❒ 为什么写这个程序（它的用途）
- ❒ 这个程序是谁写的
- ❒ 这个程序面向什么人（它的用户）
- ❒ 如何组织

更大、更复杂的程序往往有更多文档。

第 6 章中提到的 Python 帮助就是一种文档。它的作用是帮助用户了解 Python 如何工作。

9.2　单行注释

在任何代码行前面加上“#”符号就可以把它变成一个注释。（这个符号叫做数字符号，有时也叫做镑符号。）

```
# 这是 Python 程序中的一个注释
print 'This is not a comment'
```

如果运行这两行代码，会得到下面的输出：

```
This is not a comment
```

程序运行时第一行会被忽略。注释（以 # 字符开头的代码行）只是用来方便你和其他人读懂代码的。

9.3　行末注释

还可以在一行代码的最后加注释，像下面这样：

```
area = length * width # 计算矩形的面积
```

注释从 # 字符开始。# 之前的所有内容都是正常的代码行，在它后面的所有内容则是注释。

9.4　多行注释

有时你可能想使用多行注释。可以使用多行，每行前面都有一个 # 字符，像下面这样：

```
# ***************
# 这个程序用来说明 Python 中如何使用注释
# 星号所在的行只为将注释
# 与其余代码清楚地区分开
# ***************
```

多行注释可以很好地“突出”代码段，使你读代码时能清楚地区分不同代码段。可以用多行注释来描述一段代码要做什么。程序最开始的多行注释可以列出作者的名字、程序名、编写或更新的日期，以及你认为可能有用的任何其他信息。

三重引号字符串

Python 中还有一种方法可以相当于多行注释。只需建立一个没有名字的三重引

号字符串。还记得在第 2 章中曾经说过，三重引号字符串是一个可以跨多行的字符串。所以可以这样写：

```
""" 这是一个包括多行的注释，
使用了三重引号字符串。
这不完全是注释，不过也可以
相当于注释。
"""
```

因为这个字符串没有名字，而且程序对这个字符串不“做”任何处理，所以它对程序的运行没有任何影响。它相当于一个注释，尽管从严格的 Python 术语来讲这并不是一个真正的注释。



像（Python）程序员一样思考

有些 Python 程序员认为不应该使用三重引号字符串（多行字符串）作为注释。就我个人来说，我看不出这有什么充分的理由。加注释的目的就是让你的代码更易读、更容易理解。如果你觉得三重引号字符串很方便，可能会更愿意在代码中加入注释，这毕竟是件好事。

如果在 IDLE 编辑器键入一些注释，可以看到注释会用不同的颜色显示。这是为了帮助你更容易地读代码。

大多数代码编辑器允许你改变注释的颜色（或者可以改变代码其他部分的颜色）。IDLE 中注释的默认颜色是红色。因为三重引号字符串不是真正的 Python 注释，它们的颜色会不同。在 IDLE 中三重引号字符串是绿色，因为绿色是 IDLE 中字符串的默认颜色。

9.5 注释风格

现在你已经知道了如何加注释。但是应该向注释里放什么内容呢？因为它们并不影响程序如何运行，我们说注释只是一个“风格”问题。这说明，可以在注释中放你想放的任何东西（也可以根本不使用注释）。不过这并不表示注释不重要。大多数程序员都是费了一番周折才领悟到这一点。他们回头看几年前、几个月前或者是几个星期前，甚至只是昨天才写的程序时，可能完全看不明白，这往往因为他们没有加入足够的注释来解释程序是如何工作的。此时他们就会深深体会到注释的重要

性！尽管写程序时很明白，但是等以后再看程序时，很可能对原先的想法一头雾水。



至于应该在注释中放什么内容并没有严格的规定，不过建议你尽可能增加注释。现在看来，注释越多越好。宁可过于谨慎注释过多，也比注释过少要好。积累更多的编程经验后，你就会慢慢了解加多少注释以及加哪种注释最适合了。

本书中的注释

在你手上的这本书中，你可能看不到代码清单中有多少注释。这是因为这本书使用了“注解”，也就是代码旁边的说明。不过如果查看 \examples 文件夹中或者网站上的代码清单，你会看到所有代码中都有注释。

9.6　注释掉

还可以使用注释临时跳过程序中的某些部分。作为注释的所有内容都会被忽略。

```
#print "Hello"
print "World"

>>> =============== RESTART ================
>>>
World
>>>
```

由于 `print "Hello"` 被注释掉，所以这一行不会执行，不会打印“Hello”。

调试程序时，如果只希望某些部分运行而将另外的部分忽略，这会很有用。如果你希望计算机忽略某些代码行，只需要在那些代码行前面加一个 #，或者在这段代码前后加上三重引号。

大多数代码编辑器（包括 IDLE）都有一个特性，允许你很快地注释整个代码块（也能很快地取消注释）。要做到这一点，在 IDLE 的编辑器中要看 Format 菜单。

你学到了什么

在这一章，你学到了以下内容。

- 注释只是为了方便你（和其他人），而不是用来帮助计算机。
- 注释还可以用来隔离部分代码，不让它们运行。
- 可以使用三重引号字符串作为一种跨多行的注释。

测试题

由于注释相当简单，所以我们可以休息一下，这一章没有测验题。

动手试一试

再来看第 3 章“动手试一试”中的温度转换程序，增加一些注释。重新运行程序，看看运行结果是不是还一样。

第 10 章

游戏时间到了

学习编程有一种惯常的做法，就是先键入一些代码，尽管你可能完全不理解这些代码。确实是这样！

有时仅仅键入代码就能让你对程序如何工作找到一点“感觉”，虽然并不是每一行或每一个关键字都理解。我们在第 1 章就是这么做的，就是那个猜数游戏。现在还是用这个老办法建立一个程序，不过这个程序更长也更有意思。

Skier

Skier（滑雪的人）是一个非常简单的滑雪游戏，灵感来自一个名叫 SkiFree 的游戏。（你可以在 en.wikipedia.org/wiki/SkiFree 找到有关 SkiFree 的所有信息。）

在这个游戏中，你要滑下小山，努力避开树而且要尽量捡起小旗。捡起一个小旗得 10 分；碰到树则会丢掉 100 分。

运行这个程序时，会看到如右图所示的场景：



Skier 使用一个名叫 Pygame 的模块来帮助实现图形。Pygame 是一个 Python 模块（module）（我们会在第 15 章更多地讨论模块）。如果你运行了这本书的安装程序，那就已经安装了 Pygame。如果尚未安装，可以从 www.pygame.org 下载。我们会在第 16 章学习有关 Pygame 的内容。

这个程序需要如下一些图形文件：

- skier_down.png　　skier_right1.png
- skier_crash.png　　skier_right2.png
- skier_tree.png　　skier_left1.png
- skier_flag.png　　skier_left2.png

可以在 \examples\skier 文件夹找到这些文件（如果运行过安装程序），或者在本书的网站上也可以找到这些图形文件。要把它们放在保存程序的同一个文件夹或目录中，这一点非常重要。如果它们与程序不在同一个目录下，Python 就无法找到这些文件，这个程序也将无法正常工作。

Skier 的代码见代码清单 10-1。这个代码清单有点长，大约 100 行代码（为了方便阅读，这里还加入了一些空行），不过建议你还是花点时间自己亲手键入这些代码。代码清单中有一些说明，解释了代码所做的工作。注意，在代码中看到 `__init__` 时，`init` 的两边各有两条下划线。也就是说，`init` 之前和之后都有两条下划线，而不是一边一条。

代码清单 10-1　Skier

```
import pygame, sys, random

skier_images = ["skier_down.png", "skier_right1.png",
                "skier_right2.png", "skier_left2.png",
                "skier_left1.png"]

class SkierClass(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("skier_down.png")
        self.rect = self.image.get_rect()
        self.rect.center = [320, 100]
        self.angle = 0
```

创建滑雪者

```
    def turn(self, direction):
        self.angle = self.angle + direction
        if self.angle < -2:  self.angle = -2
        if self.angle >  2:  self.angle =  2
        center = self.rect.center
        self.image = pygame.image.load(skier_images[self.angle])
        self.rect = self.image.get_rect()
        self.rect.center = center
        speed = [self.angle, 6 - abs(self.angle) * 2]
        return speed
    def move(self, speed):
        self.rect.centerx = self.rect.centerx + speed[0]
        if self.rect.centerx < 20:  self.rect.centerx = 20
        if self.rect.centerx > 620: self.rect.centerx = 620

class ObstacleClass(pygame.sprite.Sprite):
    def __init__(self, image_file, location, type):
        pygame.sprite.Sprite.__init__(self)
        self.image_file = image_file
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.center = location
        self.type = type
        self.passed = False
    def update(self):
        global speed
        self.rect.centery -= speed[1]
        if self.rect.centery < -32:
            self.kill()

def create_map():
    global obstacles
    locations = []
    for i in range(10):
        row = random.randint(0, 9)
        col = random.randint(0, 9)
        location  = [col * 64 + 20, row * 64 + 20 + 640]
        if not (location in locations):
            locations.append(location)
            type = random.choice(["tree", "flag"])
            if type == "tree": img = "skier_tree.png"
            elif type == "flag":  img = "skier_flag.png"
            obstacle = ObstacleClass(img, location, type)
            obstacles.add(obstacle)

def animate():
    screen.fill([255, 255, 255])
    obstacles.draw(screen)
    screen.blit(skier.image, skier.rect)
    screen.blit(score_text, [10, 10])
    pygame.display.flip()
```

滑雪者转向

滑雪者左右移动

创建树和小旗

让场景向上滚

删除从屏幕上方滚下的障碍物

创建一个窗口，包含随机的树和小旗

重绘屏幕

```
pygame.init()
screen = pygame.display.set_mode([640,640])
clock = pygame.time.Clock()
skier = SkierClass()
speed = [0, 6]
obstacles = pygame.sprite.Group()
map_position = 0
points = 0
create_map()
font = pygame.font.Font(None, 50)

running = True
while running:

clock.tick(30)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                speed = skier.turn(-1)
            elif event.key == pygame.K_RIGHT:
                speed = skier.turn(1)
    skier.move(speed)

    map_position += speed[1]

    if map_position >=640:
        create_map()
        map_position = 0

    hit =  pygame.sprite.spritecollide(skier, obstacles, False)
    if hit:
        if hit[0].type == "tree" and not hit[0].passed:
            points = points - 100
            skier.image = pygame.image.load("skier_crash.png")
            animate()
            pygame.time.delay(1000)
            skier.image = pygame.image.load("skier_down.png")
            skier.angle = 0
            speed = [0, 6]
            hit[0].passed = True
        elif hit[0].type == "flag" and not hit[0].passed:
            points += 10
            hit[0].kill()

    obstacles.update()
    score_text = font.render("Score: " +str(points), 1, (0, 0, 0))
    animate()
pygame.quit()
```

做好准备

开始主循环

每秒更新 30 次图形

检查按键或窗口是否关闭

移动滑雪者

滚动场景

创建一个新窗口，包含场景

检查是否碰到树或得到小旗

显示得分

代码清单 10-1 的代码已经放在 \examples\skier 文件夹中，所以如果你键入的程序无法执行，或者不想完全自己键入，也可以使用这个文件。不过不管你是否相信，与简单地打开和查看代码清单相比，亲手键入这些代码会让你有更多收获。

在后面的几章，我们将会学习用于 Skier 中的所有关键字和技术。本书第 25 章将用一整章来详细解释 Skier 程序的工作原理。但是现在，你只需要键入这个程序，试着运行看看。

动手试一试

这一章你要做的只是键入这个 Skier 程序（代码清单 10-1），再运行试试看。如果运行时遇到错误，看看错误消息，试着找出错误究竟出现在哪里。

祝你好运！

第 11 章

嵌套与可变循环

我们已经看到了，在循环体（也就是代码块）中可以放入其他代码，这些代码本身也可以有自己的代码块。如果查看第 1 章中的猜数程序，可以看到：

```
while guess != secret and tries < 6:
    guess = input("What's yer guess? ")
    if guess < secret:
        print "Too low, ye scurvy dog!"
    elif guess > secret:
        print "Too high, landlubber!"
    tries = tries + 1
```

while 循环块

if 块

elif 块

外层浅灰色的块是一个 `while` 循环块，深灰色的块是这个 `while` 循环块中的 `if` 和 `elif` 块。

还可以把一个循环放在另一个循环中。这些循环就叫做嵌套循环（nested loop）。

11.1 嵌套循环

还记得第 8 章“动手试一试”中你写的乘法表程序吗？如果不考虑用户输入部分，代码会是这样：

```
multiplier = 5
for i in range (1, 11):
    print i, "x", multiplier, "=", i * multiplier
```

如果想一次打印 3 个乘法表呢？这种事情正是嵌套循环最擅长的。嵌套循环就是一个循环出现在另一个循环里。对于外循环的每次迭代，内循环都要完成它的所有迭代。

要打印 3 个乘法表，只需要把原来的循环（打印一个乘法表）包含在一个外循环中（运行 3 次）。这样，程序就会打印 3 个乘法表而不只是一个。代码清单 11-1 显示了相应的代码。

代码清单 11-1　一次打印 3 个乘法表

```
for multiplier in range (5, 8):
    for i in range (1, 11):
        print i, "x", multiplier, "=", i * multiplier
    print
```

内循环打印一个乘法表

外循环分别用值 5, 6, 7 做 3 次迭代

注意必须将内循环缩进，而且 print 语句距外部 for 循环开始位置还要多加 4 个空格。这个程序会分别打印 5、6 和 7 的乘法表，每个表分别从 1 乘到 10：

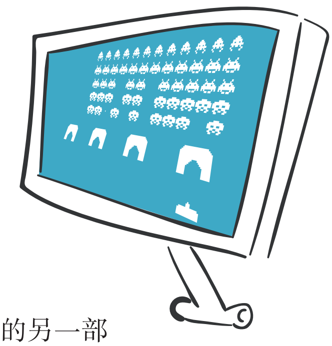
```
>>> ==================== RESTART ====================
>>>
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
4 x 5 = 20
5 x 5 = 25
6 x 5 = 30
7 x 5 = 35
8 x 5 = 40
9 x 5 = 45
10 x 5 = 50

1 x 6 = 6
2 x 6 = 12
3 x 6 = 18
4 x 6 = 24
5 x 6 = 30
6 x 6 = 36
7 x 6 = 42
8 x 6 = 48
9 x 6 = 54
10 x 6 = 60

1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
```

可以在屏幕上打印一些星号，并统计有多少个，你可能认为这很没意思，不过要了解嵌套循环到底是怎么回事，这确实是一个很好的办法。在下一节，我们就来完成这个工作。

11.2　可变循环



固定的数（比如 range() 函数中使用的数）也称为常数（constant）。如果在一个 for 循环的 range() 函数中使用常数，程序运行时循环总会运行相同的次数。在这种情况下，我们称循环次数是硬编码的（hard-coded），因为它在你的代码中被定义了，而且永远不会改变。这往往不是我们真正想要的。

有时我们希望循环次数由用户来决定，或者由程序的另一部分决定。对于这种情况，我们就需要一个变量。

例如，假设你要建立一个太空神枪手游戏。只要有外星人被消灭就要重绘屏幕。必须有某个计数器来跟踪还剩下多少外星人，另外只要屏幕更新，就需要循环处理剩下的外星人，在屏幕上画出他们的图像。每次玩家消灭一个外星人时外星人数就会改变。

因为我们还没有学习如何在屏幕上画外星人，下面先给出一个使用可变循环的简单示例程序：

```
for i in range (1, numStars):
    print '*',
```

```
>>> ===================== RESTART =====================
>>>
How many stars do you want? 5
* * * *
```

这个程序会询问用户想要多少个星号，然后使用一个可变循环准确地打印这些星号。嗯，只能算基本准确！我们想要 5 个星号，可是只得到了 4 个！唉呀，我们忘记了 for 循环不是达到 range 函数中第二个数时才停止，它在比这个数少 1 时就停止了。所以需要对用户的输入加 1。

```
numStars = int(raw_input ("How many stars do you want? "))
for i in range(1, numStars + 1):
    print '*',
```

加 1，所以如果他要 5 个星号，就会得到 5 个。

还有一种方法可以完成同样的工作，就是从 0 开始循环计数，而不是 1。（这一点在第 8 章提到过。）这种做法在编程中很常用，下一章会解释为什么。先来看看这个循环是怎样的：

```
numStars = int(raw_input ("How many stars do you want? "))
for i in range(0, numStars):
     print '*',

>>> ====================== RESTART ======================
>>>
How many stars do you want? 5
* * * * *
```

11.3　可变嵌套循环

现在来尝试一个可变嵌套循环。这就是一个嵌套循环，只不过其中一个或多个循环在 range() 函数中使用了变量。代码清单 11-2 给出了一个例子。

代码清单 11-2　一个可变嵌套循环

```
numLines = int(raw_input ('How many lines of stars do you want? '))
numStars = int(raw_input ('How many stars per line? '))
for line in range(0, numLines):
    for star in range(0, numStars):
        print '*',
    print
```

运行这个程序来看它的作用，你会看到类似这样的结果：

```
>>> ============================ RESTART ============================
>>>
How many lines of stars do you want?  3
How many stars per line?  5
*****
*****
*****
```

前两行询问用户想要多少行，以及每行希望有多少个星号。程序使用变量 numLines 和 numStars 记住这些答案。接下来有两个循环：

- 内循环（for star in range (0, numStars):）打印每个星号，对每一行上的每个星号分别运行一次；
- 外循环（for line in range (0, numLines):）对每行星号分别运行一次。

需要用第二个 print 命令开始新的一行星号。如果没有这个命令，由于第一个 print 语句中有逗号，所有星号都会打印到同一行上。

甚至可以有“嵌套嵌套循环”（或双重嵌套循环），就像代码清单 11-3 这样。

代码清单 11-3 利用双重嵌套循环生成星号块

```
numBlocks = int(raw_input ('How many blocks of stars do you want? '))
numLines = int(raw_input ('How many lines in each block? '))
numStars = int(raw_input ('How many stars per line? '))
for block in range(0, numBlocks):
    for line in range(0, numLines):
        for star in range(0, numStars):
            print '*',
        print
print
```

会得到下面的输出：

```
>>> ======================= RESTART =======================
>>>
How many blocks of stars do you want? 3
How many lines of stars in each block? 4
How many stars per line? 8
* * * * * * * *
* * * * * * * *
* * * * * * * *
* * * * * * * *

* * * * * * * *
* * * * * * * *
* * * * * * * *
* * * * * * * *

* * * * * * * *
* * * * * * * *
* * * * * * * *
* * * * * * * *
```

我们称这个循环嵌套“深度为 3”。

11.4 更多可变嵌套循环

代码清单 11-4 是代码清单 11-3 的一个更复杂的版本。

代码清单 11-4 更复杂的星号块

```
numBlocks = int(raw_input('How many blocks of stars do you want? '))
for block in range(1, numBlocks + 1):
    for line in range(1, block * 2 ):
        for star in range(1, (block + line) * 2):
            print '*',
        print
    print
```

行数和星号数的公式

输出如下：

```
>>> ======================= RESTART =======================
>>>
How many blocks of stars do you want? 3
* * * *

* * * * *
* * * * * * *
* * * * * * * * *

* * * * * * *
* * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * * * *
* * * * * * * * * * * * * * *
```

代码清单 11-4 中，外循环的循环变量用来为内循环设置范围。所以每个星号块不再有相同的行数，而且每一行也不再有相同的星号数，每次循环时行数和星号数都不同。

你希望循环嵌套多深，就可以有多深。要明白这样的嵌套循环会让人很头疼，所以有时打印出循环变量的值会很有帮助，如代码清单 11-5 所示。

代码清单 11-5　在嵌套循环中打印循环变量

```
numBlocks = int(raw_input('How many blocks of stars do you want? '))
for block in range(1, numBlocks + 1):
    print 'block = ', block
    for line in range(1, block * 2 ):
        for star in range(1, (block + line) * 2):
            print '*',
        print '  line = ', line, 'star = ', star
    print
```

显示变量

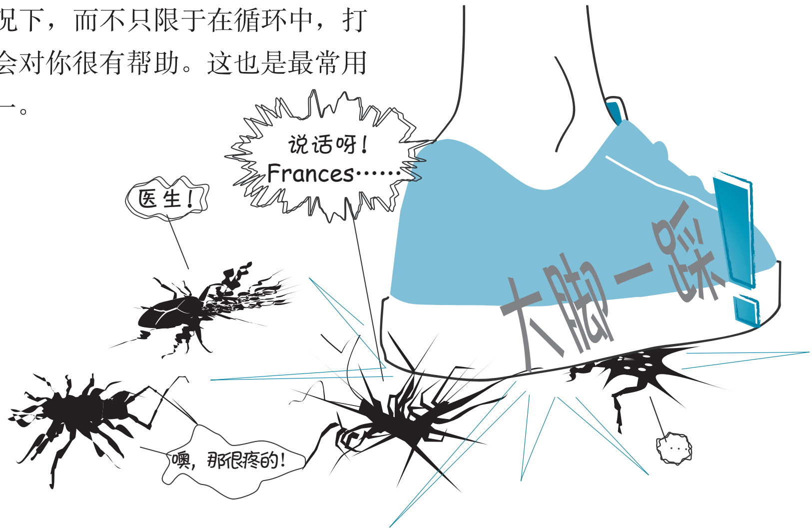
以下是这个程序的输出：

```
>>> ======================= RESTART =======================
>>>
How many blocks of stars do you want? 3
block =  1
* * *   line =  1  star = 3

block =  2
* * * * *   line =  1  star = 5
* * * * * * *   line =  2  star = 7
* * * * * * * * *   line =  3  star = 9

block =  3
* * * * * * *   line =  1  star = 7
* * * * * * * * *   line =  2  star = 9
* * * * * * * * * * *   line =  3  star = 11
* * * * * * * * * * * * *   line =  4  star = 13
* * * * * * * * * * * * * * *   line =  5  star = 15
```

在很多情况下，而不只限于在循环中，打印变量的值都会对你很有帮助。这也是最常用的调试方法之一。



11.5 使用嵌套循环

那么我们能够用嵌套循环做些什么呢？嗯，嵌套循环最擅长的工作就是得出一系列决定的所有可能的排列和组合。

术语箱

排列（permutation）是一个数学概念，表示结合一组事物的唯一方式。组合（combination）与它很类似。它们的区别在于，对于组合，顺序并不重要，而排列中顺序会有影响。

如果要从 1 到 20 选择 3 个数，可以选择

- 5, 8, 14
- 2, 12, 20

等等。如果想建立一个列表，列出从 1 到 20 选择 3 个数的所有排列，下面这两项是不同的：

- 5, 8, 14
- 8, 5, 14

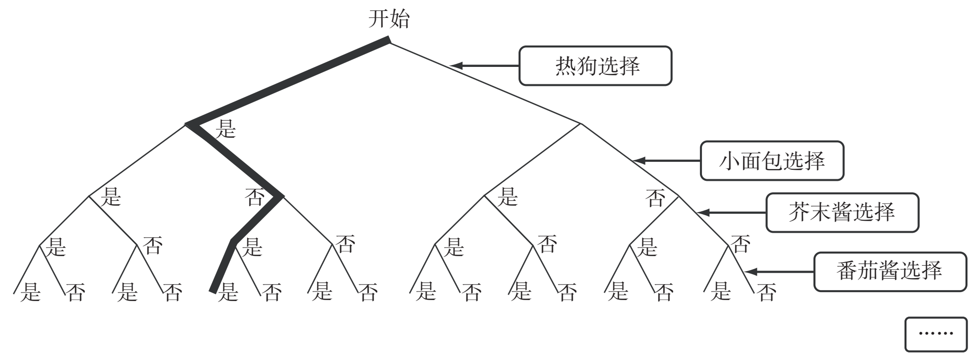
这是因为，对于排列，元素出现的顺序很重要。如果建立一个包含所有组合的列表，下面这些都会视为一项：

- 5, 8, 14
- 8, 5, 14
- 8, 14, 5

这是因为对于组合来说，顺序并不重要。

要解释这个问题，最好的办法就是举一个例子。下面假设你要在学校开春季交易会期间开个热狗店，你想做个广告海报，用数字显示如何订购热狗、小面包、番茄酱、芥末酱和洋葱的所有可能的组合。所以我们需要得出总共有多少种可能的组合。

考虑这个问题的一种方法就是使用决策树（decision tree）。下面的图显示了这个热狗问题的决策树。



每个决策点都有两种选择，是（Y）或者否（N）。这棵树的每一条不同的路径分别描述了热狗各部分的不同的组合。这里突出显示的路径是这样选择的：热狗选择 Y，小面包选择 N，芥末酱选择 Y，番茄酱选择 Y 。

现在我们使用嵌套循环来列出所有组合，也就是这棵决策树的所有路径。由于这里有 5 个决策点，所以在我们的决策树中有 5 层，相应地，在程序中就会有 5 个嵌套循环。（上图只显示了决策树的前 4 层。）

在 IDLE 编辑器窗口中键入代码清单 11-6 中的代码，保存为 hotdog1.py。

代码清单 11-6　热狗组合

```
print "\tDog \tBun \tKetchup\tMustard\tOnions"
count = 1
for dog in [0, 1]:
    for bun in [0, 1]:
        for ketchup in [0, 1]:
            for mustard in [0, 1]:
                for onion in [0, 1]:
                    print "#", count, "\t",
                    print dog, "\t", bun, "\t", ketchup, "\t",
                    print mustard, "\t", onion
                    count = count + 1
```

热狗循环
小面包循环
番茄酱循环
芥末酱循环
洋葱循环

看到这些循环是如何一个套一个的了吗？这正是嵌套循环，即一个循环放在另一个循环中。

- 外循环（热狗循环）运行两次。
- 对热狗循环的每一次迭代，小面包循环运行两次，所以它会运行 2 × 2 = 4 次。
- 对小面包循环的每一次迭代，番茄酱循环运行两次，所以它会运行 2 × 2 × 2 = 8 次。

依此类推。

最内层循环（嵌套最深的循环，也就是洋葱循环）会运行 2 × 2 × 2 × 2 × 2 = 32 次。这就涵盖了所有可能的组合。因此共有 32 种可能的组合。

如果运行代码清单 11-6 中的程序，会得到下面的结果：

```
>>> =========================== RESTART ===========================
>>>
        Dog     Bun     Ketchup Mustard Onions
# 1     0       0       0       0       0
# 2     0       0       0       0       1
# 3     0       0       0       1       0
# 4     0       0       0       1       1
# 5     0       0       1       0       0
# 6     0       0       1       0       1
# 7     0       0       1       1       0
# 8     0       0       1       1       1
# 9     0       1       0       0       0
# 10    0       1       0       0       1
# 11    0       1       0       1       0
# 12    0       1       0       1       1
# 13    0       1       1       0       0
# 14    0       1       1       0       1
# 15    0       1       1       1       0
# 16    0       1       1       1       1
# 17    1       0       0       0       0
# 18    1       0       0       0       1
# 19    1       0       0       1       0
# 20    1       0       0       1       1
# 21    1       0       1       0       0
# 22    1       0       1       0       1
# 23    1       0       1       1       0
# 24    1       0       1       1       1
# 25    1       1       0       0       0
# 26    1       1       0       0       1
# 27    1       1       0       1       0
# 28    1       1       0       1       1
# 29    1       1       1       0       0
# 30    1       1       1       0       1
# 31    1       1       1       1       0
# 32    1       1       1       1       1
```

这 5 个嵌套循环可以得到热狗、小面包、番茄酱、芥末酱和洋葱的所有可能的组合。

代码清单 11-6 中，我们使用了制表符来实现对齐，也就是符号 \t。我们还没有讨论到打印格式，不过如果你想了解更多，可以先看看第 21 章。



这里使用了一个名为 count 的变量对各个组合编号。例如，一个带小面包和芥末酱的热狗就是 #27。当然，这 32 个组合中有些组合并没有实际意义。（如果没有小面包，但有番茄酱和芥末酱，这样的热狗肯定会弄得一团糟。）要知道有句话是这么说的："顾客就是上帝！"

计算卡路里

因为如今所有人都很关心营养问题。下面为菜单上的每个组合增加一个卡路里计算。（你可能不太关心卡路里，不过我打赌你的爸爸妈妈一定很关心！）我们可以利用这个机会使用 Python 的一些数学功能（这在第 3 章学过）。

我们已经知道了每个组合里有哪些项。现在需要的就是每一项的卡路里数。然后可以在最内层循环中把各项的卡路里数加起来。

下面的代码设置了每一项有多少卡路里：

```
dog_cal = 140
bun_cal = 120
mus_cal = 20
ket_cal = 80
onion_cal = 40
```

现在只需要把它们加起来。我们知道每个菜单组合中各项要么是 0 要么是 1。所以可以直接将数量乘以每一项的卡路里，像这样：

```
tot_cal = (dog * dog_cal) + (bun * bun_cal) + \
          (mustard * mus_cal) + (ketchup * ket_cal) + \
          (onion * onion_cal)
```



由于运算的先后顺序是先算乘法再算加法，所以这里原本不需要加括号。之所以加括号是为了更容易地看出这是怎么做的。

长代码行

注意到以上代码中行末的反斜线（\）字符了吗？如果有一个很长的语句，在一行里放不下，就可以使用反斜线字符告诉 Python，“这一行还没有结束。下一行的内容也是这一行的一部分”。这里使用了两个反斜线把一个长代码行分成了 3 个小代码行。反斜线也称为行联接符（line continuation character），很多编程语言都有这种行联接符。

还可以在整个表达式前后两边额外加一对小括号，这样不必使用反斜线也可以把语句分为多行，就像下面这样：

```
tot_cal = ((dog * dog_cal) + (bun * bun_cal) +
           (mustard * mus_cal) + (ketchup * ket_cal) +
           (onion * onion_cal))
```

综合上面的内容，增加卡路里计算的热狗程序版本如代码清单 11-7 所示。

代码清单 11-7 能计算卡路里的热狗程序

```
dog_cal = 140
bun_cal = 120
ket_cal = 80
mus_cal  = 20
onion_cal = 40

print "\tDog \tBun \tKetchup\tMustard\tOnions\tCalories"
count = 1
for dog in [0, 1]:
    for bun in [0, 1]:
        for ketchup in [0, 1]:
            for mustard in [0, 1]:
                for onion in [0, 1]:
                    total_cal = (bun * bun_cal)+(dog * dog_cal) + \
                         (ketchup * ket_cal)+(mustard * mus_cal) + \
                               (onion * onion_cal)
                    print "#", count, "\t",
                    print dog, "\t", bun, "\t", ketchup, "\t",
                    print mustard, "\t", onion,
                    print "\t", total_cal
                    count = count + 1
```

列出热狗各部分的卡路里

打印表头

热狗循环是外循环

嵌套循环

内循环中计算卡路里

在 IDLE 中运行代码清单 11-7 中的程序，应该能得到这样的输出：

```
>>> =========================== RESTART ===========================
>>>
          Dog       Bun       Ketchup  Mustard  Onions    Calories
# 1       0         0         0        0        0         0
# 2       0         0         0        0        1         40
# 3       0         0         0        1        0         20
```

```
# 4       0       0       0       1       1       60
# 5       0       0       1       0       0       80
# 6       0       0       1       0       1       120
# 7       0       0       1       1       0       100
# 8       0       0       1       1       1       140
# 9       0       1       0       0       0       120
# 10      0       1       0       0       1       160
# 11      0       1       0       1       0       140
# 12      0       1       0       1       1       180
# 13      0       1       1       0       0       200
# 14      0       1       1       0       1       240
# 15      0       1       1       1       0       220
# 16      0       1       1       1       1       260
# 17      1       0       0       0       0       140
# 18      1       0       0       0       1       180
# 19      1       0       0       1       0       160
# 20      1       0       0       1       1       200
# 21      1       0       1       0       0       220
# 22      1       0       1       0       1       260
# 23      1       0       1       1       0       240
# 24      1       0       1       1       1       280
# 25      1       1       0       0       0       260
# 26      1       1       0       0       1       300
# 27      1       1       0       1       0       280
# 28      1       1       0       1       1       320
# 29      1       1       1       0       0       340
# 30      1       1       1       0       1       380
# 31      1       1       1       1       0       360
# 32      1       1       1       1       1       400
```

想想看，如果你自己手工计算所有这些组合的卡路里该是多么枯燥，即使用计算器来完成数学运算也会很乏味。编写一个程序，让它帮你把这些都算出来就有意思多了。运用循环和 Python 中的一点数学运算，这个任务对它来说简直是小菜一碟。

你学到了什么

在这一章，你学到了以下内容。

- 嵌套循环。
- 可变循环。
- 排列和组合。
- 决策树。

测试题

1. Python 中如何建立可变循环？
2. Python 中如何建立嵌套循环？
3. 下面的代码总共会打印出多少星号：

```
for i in range(5):
    for j in range(3):
        print '*',
    print
```

4. 第 3 题中的代码会得到什么输出？
5. 如果一个决策树有 4 层，每层有两个选择，共有多少种可能的选择（决策树有多少条路径）？

动手试一试

1. 还记得第 8 章创建的倒计时定时器程序吗？在这儿呢，提醒你一下：

```
import time
for i in range (10, 0, -1):
    print i
    time.sleep(1)
print "BLAST OFF!"
```

使用一个可变循环修改程序。这个程序要询问用户向下计数应当从哪里开始，比如：

```
Countdown timer:  How many seconds?  4
4
3
2
1
BLAST OFF!
```

2. 根据第 1 题写的程序，让它除了打印各个数之外还要打印一行星号，如下：

```
Countdown timer:  How many seconds?  4
4 * * * *
3 * * *
2 * *
1 *
BLAST OFF!
```

（提示：可能需要使用一个嵌套循环。）

第 12 章

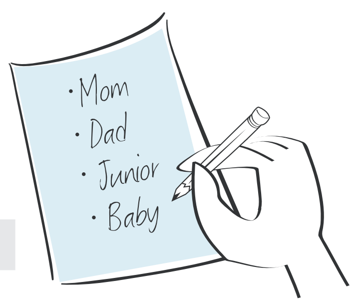
收集起来——列表与字典

我们已经见过 Python 可以在内存中存储信息，还可以用名字来获取原先存储的信息。到目前为止，我们存储过字符串和数（包括整数和浮点数）。有时候可以把一堆东西存储在一起，放在某种“组”或者“集合”中，这可能很有用。这样一来，就可以一次对整个集合做某些处理，也能更容易地记录一组东西。有一类集合叫做列表（list），另一类叫做字典（dictionary）。在这一章中，我们就来学习列表和字典的相关知识——什么是列表，如何创建、修改和使用列表。

列表非常有用，很多很多程序里都用到了列表。后面几章开始讨论图形和游戏编程时我们的例子中就会大量使用列表，因为游戏中的很多图形对象通常都存储在列表中。

12.1 什么是列表

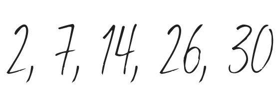
如果我让你建一个家庭成员列表，你可能会像右图这样写：



在 Python 中，就要写成：

```
family = ['Mom', 'Dad', 'Junior', 'Baby']
```

如果我让你写下你的幸运数字，你可能会这样写：



在 Python 中，就要写成：

```
luckyNumbers = [2, 7, 14, 26, 30]
```

family 和 luckyNumbers 都是 Python 列表的例子，列表中的单个元素就叫做项或者元素（item）。可以看到，Python 中的列表与你在日常生活中建立的列表并没有太大差异。列表使用中括号来指出从哪里开始，到哪里结束，另外用逗号分隔列表内的各项。

12.2 创建列表

family 和 luckyNumbers 都是变量。前面曾经说过，可以为变量赋不同类型的值。我们已经为变量赋过数和字符串，还可以为变量赋一个列表。

就像创建任何其他变量一样，创建列表也是要为它赋某个值，如前面对 luckyNumbers 的赋值。另外还可以创建一个空的列表，如下：

```
newList = []
```

中括号没有任何元素，所以这个列表是空的。不过一个空列表有什么用呢？为什么想要创建这样一个空列表呢？

嗯，很多情况下，我们无法提前知道列表中会有些什么。我们不知道其中会有多少元素，也不知道这些元素是什么，只知道将会用一个列表来保存这些内容。有了空列表后，程序就可以向这个列表中增加元素。这又怎么做到呢？

12.3 向列表增加元素

要向列表增加元素，需要使用 append()。在交互模式中试试下面的代码：

```
>>> friends = []                  ← 建立一个新的空列表
>>> friends.append('David')         ← 向列表增加一项 "David"
>>> print friends
```

你会得到这样的结果：

```
['David']
```

再来增加一个元素：

```
>>> friends.append('Mary')
>>> print friends
['David', 'Mary']
```

记住，向列表增加元素之前，必须先创建列表（可以是空列表，也可以非空）。这就像在做一个蛋糕：不能直接把各种配料倒在一起，而是先将配料倒入碗中，不然肯定会弄得到处都是！



12.4　这个点是什么

为什么要在 friends 和 append() 之间加一个点（.）呢？嗯，现在要谈到一个重要的话题了：这就是对象。我们会在第 14 章学习关于对象的更多内容，不过现在先简单解释一下。

术语箱

追加（append）是指把一个东西加在最后面。

把一个东西追加到列表时，会把它增加到列表的末尾。

Python 中的很多东西都是对象（object）。要想用对象做某种处理，需要这个对象的名字（变量名），然后是一个点，再后面是要对对象做的操作。所以要向 friends 列表追加一个元素，就要写成：

```
friends.append(something)
```

12.5　列表可以包含任何内容

列表可以包含 Python 能存储的任何类型的数据，这包括数字、字符串、对象，甚至可以包含其他列表。并不要求列表中的元素是同种类型或同一种东西。这说明，一个列表中可以同时包含不同类型，例如数字和字符串，可能像这样：

```
my_list = [5, 10, 23.76, 'Hello', myTeacher, 7, another_list]
```

下面用一些简单的内容建立一个新列表，比如字母表中的字母，这样我们在学习列表时就能更容易地了解做了些什么。在交互模式中键入下面的代码：

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
```

12.6　从列表获取元素

可以按元素的索引（index）号从列表获取单个元素。列表索引从 0 开始，所以这个列表中的第一项就是 letters[0]。

```
>>> print letters[0]
a
```

再来试一个：

```
>>> print letters[3]
d
```

为什么索引从 0 而不是 1 开始?

从计算机发明到现在，很多程序员、工程师还有计算机科学家们一直都在争论这个问题。我可不想陷入这场争论中，所以直接告诉你答案：“因为事实就是这样。”下面我们继续……



好吧，好吧，可以看看下面的“到底怎么回事”，这里解释了为什么索引从 0 而不是从 1 开始。

到底怎么回事？







你应该记得计算机使用二进制数也就是“比特”来存储一切信息。很久以前，这些比特非常贵重。每一个比特都必须精挑细选，还要靠毛驴从比特农场搬运……这只是开个玩笑。不过这些比特位确实很昂贵。

二进制计数从 0 开始。所以，为了最高效地使用比特位而没有任何浪费，内存位置和列表索引也都从 0 开始。

你很快就会习惯从 0 开始索引，因为这在编程中相当常见。

注意！这个词有意思！

索引（index）表示某个东西的位置。index 的复数形式是 indices（不过有些人也用 indexes 作为 index 的复数形式）。

如果你在队伍中排在第 4 个，你在这个队伍中的索引就是 4。不过，如果你是一个 Python 列表中的第 4 个人，索引则是 3，因为 Python 列表索引从 0 开始！

12.7 列表“分片”

还可以使用索引从列表一次获取多个元素。这叫做列表分片（slicing）。

```
>>> print letters[1:4]
['b', 'c', 'd']
```

与 for 循环中的 range() 类似，分片获取元素时，会从第一个索引开始，不过在达到第二个索引之前停止。正是因为这个原因，前面的例子中我们只取回 3 项，而不是 4 项。要记住这一点，一种方法就是牢记取回的项数总是两个索引数之差（4 – 1 = 3，所以取回 3 项）。



关于列表分片，还有一个重要的问题需要记住：对列表分片时取回的是另一个（通常更小的）列表。这个更小的列表称为原列表的一个分片（slice）。原来的列表并没有改变。这个分片是原列表的部分副本（copy）。

下面来看这有什么不同：

```
>>> print letters[1]
b
>>> print letters[1:2]
['b']
```

在第一种情况下，我们取回一个元素。在第二种情况下，取回的是包含这个元素的一个列表。这个差别很微妙，但是你必须知道。在第一种情况下，我们使用了一个索引从列表得到一个元素。第二种情况下则是使用分片记法来得到列表的一个单元素分片（只包含一个元素的分片）。

要真正了解二者的区别，可以试试这些命令：

```
>>> print type(letters[1])
<type 'str'>
>>> print type(letters[1:2])
<type 'list'>
```

这里分别显示了两个结果的类型（type），从中可以清楚地看出，前一种情况下得到了一个元素（这里是一个字符串），后一种情况下得到的是一个列表。

对列表分片时会得到一个较小的列表，这是原列表中元素的一个副本。这说明，可以修改这个分片，而原列表不会受到任何影响。

分片简写

使用分片时可以采用一些简写形式。即使采用这些简写，也不会减少太多键入。不过程序员总是很懒，所以他们会大量使用简写。我希望你知道这些简写是什么，这样当你在别人的代码中看到这些简写时就能认出来，而且明白是什么意思。这很重要，因为学习新的编程语言时（或者笼统地说，学习编程时），查看并且理解其他人的代码是一种很好的方法。

如果你想要的分片包括列表的开始部分，简写方式是使用冒号，后面是想要的元素个数，例如：

```
>>> print letters[:2]
['a', 'b']
```

注意，冒号前面没有数字。这样就会得到从列表起始位置开始一直到（但不包括）指定索引之间的所有元素。

要得到列表末尾也可以用类似的记法。

```
>>> print letters[2:]
['c', 'd', 'e']
```

使用一个后面跟冒号的数，这样可以得到从指定索引到列表末尾的所有元素。

如果没有放入任何数，而只有冒号，就可以得到整个列表：

```
>>> print letters[:]
['a', 'b', 'c', 'd', 'e']
```

应该记得吧？我说过分片就是建立原列表的副本。所以 letters[:] 会建立整个列表的副本。如果你想对列表做些修改，但是同时还想保持原来的列表不做任何改变，使用这种分片就会很方便。

12.8　修改元素

可以使用索引来修改某个列表元素：

```
>>> print letters
['a', 'b', 'c', 'd', 'e']
>>> letters[2] = 'z'
>>> print letters
['a', 'b', 'z', 'd', 'e']
```

但是不能使用索引向列表增加新的元素。目前，这个列表中有 5 项，索引分别是从 0 到 4。

所以不能这样做：

```
letters[5] = 'f'
```

这是不行的。（如果你愿意也可以试试看。）这就像是想要改变一个还不存在的东西。要向列表中增加元素，必须另想其他办法，我们下面就会做这个工作。不过，在此之前，先把列表改回到原来的样子：

```
>>> letters[2] = 'c'
>>> print letters
['a', 'b', 'c', 'd', 'e']
```

12.9　向列表增加元素的其他方法

我们已经看到了如何使用 append() 向列表增加元素。不过除此以外还有其他一些方法。实际上，向列表增加元素共有 3 种方法：append()、extend() 和 insert()。

- append() 向列表末尾增加一个元素。
- extend() 向列表末尾增加多个元素。
- insert() 在列表中的某个位置增加一个元素，不一定非得在列表末尾。你可以告诉它要在哪里增加元素。

增加到列表末尾：append()

我们已经见过 append() 是如何工作的。它把一个元素增加到列表末尾：

```
>>> letters.append('n')
>>> print letters
['a', 'b', 'c', 'd', 'e', 'n']
```

再来增加一项：

```
>>> letters.append('g')
>>> print letters
['a', 'b', 'c', 'd', 'e', 'n', 'g']
```

注意这些字母并没有按顺序排列。这是因为 append() 只是将元素增加到列表末尾。如果希望这些元素按顺序排列，必须对它们排序。稍后就会谈到排序。

扩展列表：extend()

extend() 在列表末尾增加多个元素：

```
>>> letters.extend(['p', 'q', 'r'])
>>> print letters
['a', 'b', 'c', 'd', 'e', 'n', 'g', 'p', 'q', 'r']
```

注意 extend() 方法的圆括号中是一个列表。列表有一个中括号，所以对于 extend()，可以同时有圆括号和中括号。

提供给 extend() 的列表中的所有内容都会增加到原列表的末尾。

插入一个元素：insert()

insert() 会在列表中的某个位置增加一个元素。可以指定希望将元素增加到列表的哪个位置：

```
>>> letters.insert(2, 'z')
>>> print letters
['a', 'b', 'z', 'c', 'd', 'e', 'n', 'g', 'p', 'q', 'r']
```

在这里，我们将字母 z 增加到索引为 2 的位置。索引 2 是列表中的第 3 个位置（因为索引从 0 开始）。原先位于第 3 个位置上的字母（也就是 c）会向后推一个位置，移到第 4 个位置上。它后面的每一个元素也都要向后移一个位置。

append() 和 extend() 的区别

有时 append() 和 extend() 看起来很类似，不过它们确实有一些区别。下面再回到原来的列表。首先，用 extend() 增加 3 个元素：

```
>>> letters = ['a','b','c','d','e']
>>> letters.extend(['f', 'g', 'h'])
>>> print letters
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

现在，再用 append() 做同样的事情：

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
>>> letters.append(['f', 'g', 'h'])
>>> print letters
['a', 'b', 'c', 'd', 'e', ['f', 'g', 'h']]
```

怎么回事？嗯，我们前面说过，append() 会向列表增加一个元素。它怎么会增加 3 个元素呢？其实它并没有增加 3 个元素，这里确实只增加了一个元素，只不过这刚好是一个包含 3 项的列表。正是这个原因，所以在这个列表中多了一对中括号。要记住，列表可以包含任何东西，也包括其他列表。这个例子就属于这种情况。

insert() 的工作与 append() 相同，只不过你可以告诉它在哪里放入新的元素。append() 总是把新元素放在列表末尾。

12.10　从列表删除元素

如何从列表删除或者去除元素呢？有 3 种方法：remove()、del 和 pop()。

用 remove() 删除元素

remove() 会从列表中删除你选择的元素，把它丢掉：

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
>>> letters.remove('c')
>>> print letters
['a', 'b', 'd', 'e']
```

你不需要知道这个元素在列表中的具体位置，只需要知道它确实在列表中（可以是任何位置）。如果你想删除的东西根本不在列表中，就会得到错误消息：

```
>>> letters.remove('f')
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    letters. remove('f')
ValueError: list.remove(x): x not in list
```

那么怎么才能知道列表中是否包含某个元素呢？后面就要讲到。先来看另外两种从列表中删除元素的方法。

用 del 删除

del 允许利用索引从列表中删除元素，如下所示：

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
>>> del letters[3]
>>> print letters
['a', 'b', 'c', 'e']
```

在这里，我们删除了第 4 个元素（索引 3），也就是字母 d。

用 pop() 删除元素

pop() 从列表中取出最后一个元素交给你。这说明，你可以为它指派一个名字，比如：

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
>>> lastLetter = letters.pop()
>>> print letters
['a', 'b', 'c', 'd']
>>> print lastLetter
e
```

使用 pop() 时还可以提供一个索引，如：

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
>>> second = letters.pop(1)
>>> print second
b
>>> print letters
['a', 'c', 'd', 'e']
```

在这里我们弹出了第 2 个字母（索引 1），也就是 b。弹出的元素赋给 second，而且会从 letters 删除。

括号里没有提供参数时，pop() 会返回最后一个元素，并把它从列表中删除。如果在括号里放入一个数，pop(n) 会给出这个索引位置上的元素，而且会把它从列表中删除。

12.11 搜索列表

列表中有多个元素时，怎么查找这些元素呢？对列表通常有两种处理：

- 查找元素是否在列表中；
- 查找元素在列表中的哪个位置（元素的索引）。

in 关键字

要找出某个元素是否在列表中，可以使用 in 关键字，例如：

```
if 'a' in letters:
    print "found 'a' in letters"
else:
    print "didn't find 'a' in letters"
```

'a' in letters 部分是一个布尔或逻辑表达式。如果 a 在这个列表中，它会返回值 True，否则返回 False。

术语箱

布尔（boolean）是一种只使用两个值（1 和 0，或者 true 和 false）的算术运算。这是数学家乔治·布尔发明的，用 and、or 和 not 来结合 true 和 false 条件（由 1 和 0 表示）时，就会用到布尔运算，我们在第 7 章中已经见过。

可以在交互模式中试试下面的命令：

```
>>> 'a' in letters
True
>>> 's' in letters
False
```

可以看到，名为 letters 的列表中确实包含一个元素 a，但是不包含元素 s。所以 a 在列表中，而 s 不在列表中。现在可以结合使用 in 和 remove() 编写一些代码，保证即使值不在列表中也不会给出错误：

```
if 'a' in letters:
    letters.remove('a')
```

查找索引

为了找出一个元素位于列表中的什么位置，可以使用 index() 方法，如下：

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
>>> print letters.index('d')
3
```

所以我们知道 d 的索引是 3，这说明它是列表中的第 4 个元素。

就像 remove() 一样，如果在列表中没有找到这个值，index() 会给出一个错误，所以最好结合使用 in，就像这样：

```
if 'd' in letters:
    print letters.index('d')
```

12.12　循环处理列表

最早开始讨论循环时，我们看到循环完成了一个值列表的迭代处理。我们还了解了 range() 函数，并用它作为快捷方式为循环生成数字列表。前面已经看到 range() 确实可以提供一个数字列表。

不过循环完全可以迭代处理任何列表，而不一定非得是数字列表。假设要显示出我们的字母列表，一行显示一个元素，可以这样做：

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
>>> for letter in letters:
```

```
    print letter
a
b
c
d
e
```

这里我们的循环变量是 letter。（之前我们使用了 looper 或 i、j 和 k 之类的循环变量。）循环迭代处理（循环处理）列表中的所有值，每次迭代时，当前元素会存储在循环变量 letter 中，然后显示出来。

12.13　列表排序

列表是一种有顺序（ordered）的集合 。这说明列表中的元素有某种顺序，每个元素都有一个位置，也就是它的索引。一旦以某种顺序将元素放在列表中，它们就会保持这种顺序，除非用 insert()、append()、remove() 或 pop() 改变列表。不过这个顺序可能不是你真正想要的顺序。你可能希望列表在使用前已经排序。

要对列表排序，可以使用 sort() 方法。

```
>>> letters = ['d', 'a', 'e', 'c', 'b']
>>> print letters
['d', 'a', 'e', 'c', 'b']
>>> letters.sort()
>>> print letters
['a', 'b', 'c', 'd', 'e']
```

sort() 会自动按字母顺序对字符串从小到大排序，如果是数字，就会按数字顺序从小到大排序。

有一点很重要，你要知道 sort() 会在原地修改列表。这说明它会改变你提供的原始列表，而不是创建一个新的有序列表。所以，你不能这样做：

```
>>> print letters.sort()
```

如果这样做，会得到“None”。必须分两步来完成，就像这样：

```
>>> letters.sort()
>>> print letters
```

按逆序排序

让一个列表按逆序排序有两种方法。一种方法是先按正常方式对列表排序，然后对这个有序列表完成逆置（reverse），如下：

```
>>> letters = ['d', 'a', 'e', 'c', 'b']
>>> letters.sort()
>>> print letters
['a', 'b', 'c', 'd', 'e']
>>> letters.reverse()
>>> print letters
['e', 'd', 'c', 'b', 'a']
```

在这里我们看到一个新的列表方法 reverse()，它会把列表中元素的顺序倒过来。

另一种方法是向 sort() 增加了一个参数，直接让它按降序排序（从大到小）：

```
>>> letters = ['d', 'a', 'e', 'c', 'b']
>>> letters.sort (reverse = True)
>>> print letters
['e', 'd', 'c', 'b', 'a']
```

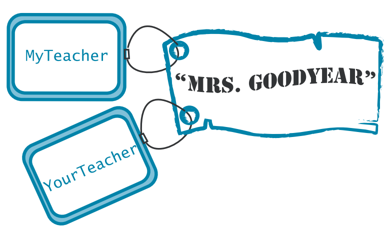
这个参数名为 reverse，它会按照你的意愿，将列表按逆序排序。

要记住，我们刚才讨论的所有排序和逆置都会对原来的列表做出修改。这说明，你原来的列表已经没有了。如果希望保留原来的顺序，而对列表的副本进行排序，可以使用分片记法建立副本，也就是与原列表相等的另一个列表（有关的内容已经在这一章前面讨论过）：

```
>>> original_list = ['Tom', 'James', 'Sarah', 'Fred']
>>> new_list = original_list[:]
>>> new_list.sort()
>>> print original_list
['Tom', 'James', 'Sarah', 'Fred']
>>> print new_list
['Fred', 'James', 'Sarah', 'Tom']
```



卡特，很高兴你问这个问题。如果你还记得很早很早以前我们刚开始谈到名字和变量时（第 2 章），曾经说过，完成 name1 = name2 之类的操作时，就是为同一个东西建立一个新的名字。应该还记得这个图：

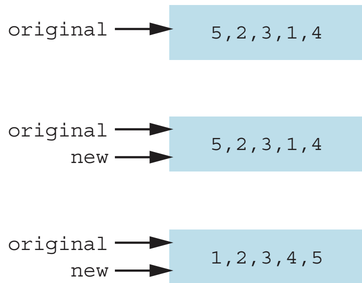


所以为一个东西指定另一个名字时，只是向同一个东西增加一个新的标签。在卡特的这个例子中，new_list 和 original_list 都表示同一个列表。可以用任何一个名字来改变列表（例如，可以对它排序）。不过，这里仍然只有一个列表，就如：

```
original = [5,2,3,1,4]

new = original

new.sort()
```



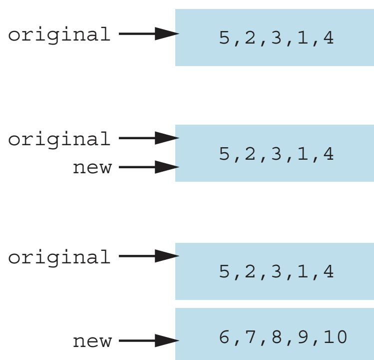
我们对 new 完成排序，但是 original 也同样得到排序，因为 new 和 original 只是同一个列表的两个不同名字。这里并没有两个不同的列表。

当然，也可以把 new 标签移到一个全新的列表上，就像这样：

```
original = [5,2,3,1,4]

new = original

new = [6,7,8,9,10]
```

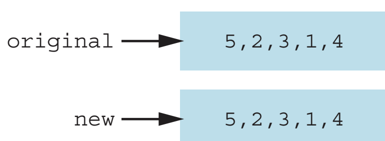


第 2 章对字符串和数就是这样做的。

这说明，如果你确实想建立一个列表的副本，就要另想办法，而不能只是用 new = original。要达到这个目的，最容易的方法是使用分片记法，就像前面所做的：new = original[:]。这表示“复制列表中的所有内容，从第一个元素到最后一个元素”。这样就可以得到：

```
original = [5,2,3,1,4]

new = original [:]
```



这里有两个不同的列表。我们建立了原列表的副本，命名为 new。现在如果对一个列表排序，另一个列表将不会同时排序。

另一种排序方法——sorted()

还有一种方法可以得到一个列表的有序副本而不会影响原列表的顺序。Python 提供了一个名为 sorted() 的函数可以完成这个功能。它的工作如下：

```
>>> original = [5, 2, 3, 1, 4]
>>> newer = sorted(original)
>>> print original
[5, 2, 3, 1, 4]
>>> print newer
[1, 2, 3, 4, 5]
```

sorted() 函数提供了原列表的一个有序副本。

12.14 可改变和不可改变

如果还记得第 2 章中的内容，我们说过，真正改变一个数或字符串是做不到的，你能改变的只是把一个名字指派到哪个数或字符串（换句话说，你只能移动标签）。不过，Python 中确实有一些可以改变的类型，列表就是其中之一。刚才已经看到，列表可以追加或删除元素，另外列表中的元素还可以排序或逆置。

这两种不同的变量分别称为可改变和不可改变的变量。可改变（mutable）是指“能够改变”或者“可以改变”。不可改变（immutable）表示“不能改变”或者“不可以改变”。在 Python 中，数字和字符串是不可改变的（不能改变），而列表是可改变的（能够改变）。

元组——不可改变的列表

有些情况下你可能不希望列表可以改变。Python 中有没有一种不可改变的列表呢？答案是肯定的。确实有一个名为元组（tuple）的类型，这就属于不可改变的列表。可以这样来建立元组：

```
my_tuple = ("red", "green", "blue")
```

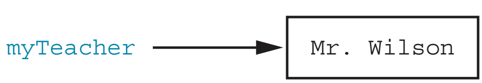
这里使用了圆括号，而不是列表使用的中括号。

由于元组是不可改变的，所以不能对元组完成排序，也不能追加和删除元素。一旦用一组元素创建一个元组，它就会一直保持不变。

12.15 双重列表：数据表

考虑数据如何存储在程序中时，可以用图直观地表示，这很有用。

变量有一个值。



列表就像是把一行值串在一起。

myFriends →

Curtis	Karla	Jenn	Kim	Shaun

有时还需要一个包含行和列的表。

classMarks →

	Math	Science	Reading	Spelling
Joe	55	63	77	81
Tom	65	61	67	72
Beth	97	95	92	88

如何保存数据表呢？我们已经知道，列表中包含多个元素，可以把每个学生的成绩放在一个列表中，像这样：

```
>>> joeMarks = [55, 63, 77, 81]
>>> tomMarks = [65, 61, 67, 72]
>>> bethMarks = [97, 95, 92, 88]
```

或者对应每个课程使用一个列表，如下：

```
>>> mathMarks = [55, 65, 97]
>>> scienceMarks = [63, 61, 95]
>>> readingMarks = [77, 67, 92]
>>> spellingMarks = [81, 72, 88]
```

不过我们可能希望把所有数据都收集到一个数据结构中。

术语箱

数据结构（data structure）是一种在程序中收集、存储或表示数据的方法。数据结构包括变量、列表和其他一些我们还没有讨论到的内容。实际上，数据结构这个词就表示程序中数据的组织方式。

要为我们的成绩建立一个数据结构，可以这样做：

```
>>> classMarks = [joeMarks, tomMarks, bethMarks]
>>> print classMarks
[[55, 63, 77, 81], [65, 61, 67, 72], [97, 95, 92, 88]]
```

这会得到一个元素列表，其中每个元素本身又是一个列表。我们创建了一个“列表的列表”（list of list），也就是双重列表。classMarks 列表中的每个元素本身又都是一个列表。

还可以直接创建 classMarks，而不需要先创建 joeMarks、tomMarks 和 bethMarks，如下：

```
>>> classMarks = [ [55,63,77,81], [65,61,67,72], [97,95,92,88] ]
>>> print classMarks
[[55, 63, 77, 81], [65, 61, 67, 72], [97, 95, 92, 88]]
```

现在来显示我们的数据结构：classMarks 有 3 个元素，每个元素分别对应一个学生。所以可以使用 in 来循环处理：

```
>>> for studentMarks in classMarks:
        print studentMarks

[55, 63, 77, 81]
[65, 61, 67, 72]
[97, 95, 92, 88]
```

这里我们对名为 classMarks 的列表完成循环处理。循环变量是 studentMarks。每次循环时，会打印列表中的一个元素。这里的每一个元素分别是一个学生的成绩，它本身也是一个列表。（前面创建过这些学生列表。）

可以注意到，这看上去与前一页的表很类似，所以我们提出的这种数据结构可以把所有数据都保存在一个地方。

从表获取一个值

怎么得到这个表（也就是双重列表）中的值呢？我们已经知道，第一个学生的成绩（joeMarks）在一个列表中，而这个列表本身是 classMarks 中的第一个元素。

下面来检查一下：

```
>>> print classMarks[0]
[55, 63, 77, 81]
```

classMarks[0] 是 Joe 的 4 门课程成绩的一个列表。现在我们想从 classMarks[0] 得到一个值。怎么做呢？可以使用第二个索引。

如果希望得到他的第三个成绩（阅读课成绩），也就是索引 2，可以这样做：

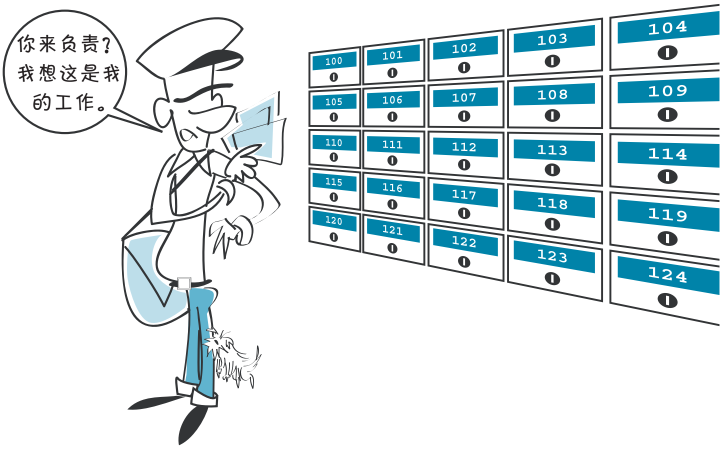
```
>>> print classMarks[0][2]
77
```

这会给出 classMarks 中的第一个元素（索引 0），也就是 Joe 的成绩列表，以及这个列表中的第三个元素（索引 2），这正是他的阅读课成绩。看到一个名字后面带着两组中括号时，比如说 classMarks[0][2]，这往往表示一个双重列表。

classMarks →

	Math	Science	Reading	Spelling
Joe	55	63	77	81
Tom	65	61	67	72
Beth	97	95	92	88

classMarks 列表并不知道 Joe、Tom 和 Beth 这些名字，也不知道数学（Math）、科学（Science）、阅读（Reading）和拼写（Spelling）这些课程。这里之所以这样标，是因为我们知道想要在这个列表中储存什么信息。不过，对于 Python 来说，它们只是列表中一些已经编号的位置而已。这就像邮局里编号的邮箱。邮箱上没有名字，只有编号。邮递员只负责明确哪封信归哪个邮箱，而你知道哪个邮箱是你的。



要对 classMarks 表加标签，一种更准确的方法应该是这样：

classMarks →	[0]	[1]	[2]	[3]
classMarks[0]	55	63	77	81
classMarks[1]	65	61	67	72
classMarks[2]	97	95	92	88

现在可以更容易地看出成绩 77 存储在 classMarks[0][2] 中。

如果编写一个程序使用 classMarks 存储我们的数据，就必须知道哪些数据存储在哪一行哪一列。就像邮递员一样，我们的任务是明确哪个位置属于哪个数据。

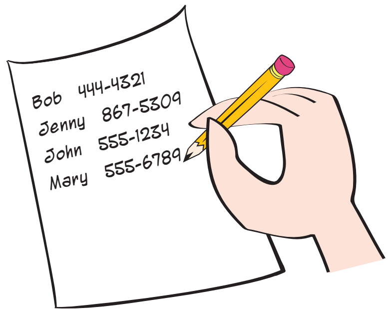
12.16　字典

从上文可以看出，Python 列表是一种将元素组织在一起的方式。在编程中，你会经常想以另一种方式组织元素，即将某个值和另一个值关联起来。这种组织方式就

像电话通讯录将姓名和电话号码关联起来，也像字典将单词和它们的含义关联起来。

Python 字典（dictionary）是一种将两个东西关联在一起的方式。被关联在一起的两个东西分别称为键（key）和值（value）。字典中的每个项（item）或条目（entry）都有一个键和一个值，它们合起来被称为键值对（key-value pair）。一个字典就是一些键值对的集合。

一个简单的例子就是电话通讯录。假设你想保存朋友们的电话号码。你会使用他们的姓名去查找他们的号码（希望你的朋友们没有重名的）。这个姓名就是“键”，即你会用它来查找信息，而电话号码就是“值”，即你要查找的信息。



下面是创建一个 Python 字典的方法，我们用它来存储姓名和电话号码。首先，创建一个空的字典：

```
>>> phoneNumbers = {}
```

这个代码看起来和创建列表的方式非常像，只不过它使用的是花括号而不是方括号。

然后，我们来添加一个条目：

```
>>> phoneNumbers["John"] = "555-1234"
```

如果我们打印一下字典，它看起来是这样的：

```
>>> print phoneNumbers
{'John': '555-1234'}
```

首先显示键，然后是一个冒号，再接着显示值。之所以用引号，是因为在这个例子中键和值刚好都是字符串（不是必需的）。

也可以用另一种方式来完成：

```
>>> phoneNumbers = {"John": "555-1234"}
```

我们来添加更多的条目。不像在列表中可以使用 `append()`，在字典中则没有用于添加新条目的方法。只需要指定新的键和值就行了：

```
>>> phoneNumbers["Mary"] = "555-6789"
>>> phoneNumbers["Bob"] = "444-4321"
>>> phoneNumbers["Jenny"] = "867-5309"
```

我们来看一下整个字典：

```
>>> print phoneNumbers
{'Bob': '444-4321', 'John': '555-1234', 'Mary': '555-6789', 'Jenny': '867-5309'}
```

我们之所以创建字典，是因为我们可以在字典中查找东西。在这个例子中，我们希望按姓名来查找电话号码。你可以这样做：

```
>>> print phoneNumbers["Mary"]
'555-6789'
```

注意，这里使用方括号来指定你想要找的条目的键。整个字典本身还是被包裹在花括号中。

字典和列表有点类似，但也有一些主要区别。这两种类型都称为“集合”（collection），也就是说，它们都是将其他类型条目集中在一起的组织方式。

它们的一些相似点：

- 列表和字典都可以包含任意类型（甚至包括列表和字典）的条目，所以你可以有一个包含数字、字符串、对象甚至其他集合的集合。
- 列表和字典都提供了在集合中查找条目的方法。

它们的一些不同点：

- 列表是有顺序的（ordered）。如果你按照某种顺序向列表中添加元素，这些元素就会保持这种顺序。你还可以对列表进行排序。字典是无序的（unordered）。如果你向字典中添加内容，然后打印出来，显示的顺序可能会跟添加的顺序不同。
- 列表中的元素是使用索引访问的，而字典中的条目是使用键来访问的：

```
>>> print myList[3]
'eggs'
>>> print myDictionary["John"]
'555-1234'
```

前面提到过，在 Python 中很多东西都是对象，列表和字典也是。所以列表和字典都有一些使用点号表示法来使用的方法。

keys() 方法会列出字典中所有的键：

```
>>> phoneNumbers.keys()
['Bob', 'John', 'Mary', 'Jenny']
```

values() 方法会列出字典中所有的值：

```
>>> phoneNumbers.values()
['444-4321', '555-1234', '555-6789', '867-5309']
```

其他的语言也有与 Python 字典类似的东西。它们通常被称为关联数组（associative array），因为它们将键和值关联在一起。你可能也会听说它们的另外一个名字——哈希表（hash table）。

和列表一样，字典中的条目也可以是任意类型，包括简单类型（整数、浮点数、字符串）和集合（列表、字典）以及复合类型（对象）。

没错，你可以在字典中包含其他的字典，正如列表中可以包含其他的列表一样。但事实上，这句话并不完全对。只有字典中的值是可以使用字典的，而键的要求更为严格一些。早先我们谈论过可变类型（mutable）与不可变类型（immutable）。字典的键只可以使用不可变类型（布尔、整数、浮点数、字符串和元组）。你不能使用一个列表或者字典来当作键，因为它们是可变类型。

我在上面提到过，字典和列表有一个不同之处，就是字典是无序的。注意，尽管 Bob 的电话是第三个被添加到字典中的，但在打印字典内容时却是第一个显示的。字典没有顺序这个概念，所以对字典进行排序是没有意义的。但有时你希望将字典中的内容按照某种顺序显示出来。虽然字典没有顺序，但可以对列表排序，所以当你拿到键的列表后，就可以对键进行排序，然后按照键的顺序显示字典内容。你可以使用 sorted() 函数来对字典的键排序，像下面这样：

```
>>> for key in sorted(phoneNumbers.keys()):
        print key, phoneNumbers[key]

Bob 444-4321
Jenny 867-5309
John 555-1234
Mary 555-6789
```

这和你在列表中看到的 sorted() 函数是一样的。如果你细想一下，会发现它是有效的，因为字典的键的集合是一个列表。

那么，如果你想将字典的值（而不是键）按某种顺序输出该怎么办呢？在电话通讯录的例子中，就是按照电话号码从小到大输出。字典的查找过程是单向的，这意味着只能用键去查找值，而不能反过来。所以对值排序会有些困难。但这仍然是

可以做到的，只是需要做更多的工作：

```
>>> for value in sorted(phoneNumbers.values()):
        for key in phoneNumbers.keys():
            if phoneNumbers[key] == value:
                print key, phoneNumbers[key]
Bob 444-4321
John 555-1234
Mary 555-6789
Jenny 867-5309
```

我们首先取得排序之后的值的列表，然后针对列表中的每个值，循环遍历字典中的所有键，直到找到与该值关联的键。

下面是字典可以做的其他一些事情。

- 使用 del 删除一个条目：

```
>>> del phoneNumbers["John"]
>>> print phoneNumbers
{'Bob': '444-4321', 'Mary': '555-6789', 'Jenny': '867-5309'
```

- 使用 clear() 删除所有条目（清空字典）：

```
>>> phoneNumbers.clear()
>>> print phoneNumbers
{}
```

- 使用 in 确定某个键在字典中是否存在：

```
>>> phoneNumbers = {'Bob':'444-4321', 'Mary':'555-6789','Jenny':'867-5309'}
>>> "Bob" in phoneNumbers
True
>>> "Barb" in phoneNumbers
False
```

字典在 Python 代码中使用得很广泛。

这些当然不是有关 Python 字典的全部内容，但你应该对其有了大致的了解，这样你就可以在代码中使用字典，也可以认出在其他代码中出现的字典。

你学到了什么

在这一章，你学到了以下内容。

- ❒ 列表是什么。
- ❒ 如何向列表中增加元素。
- ❒ 如何从列表删除元素。
- ❒ 如何确定列表是否包含某个值。
- ❒ 如何对列表排序。
- ❒ 如何建立列表的副本。
- ❒ 元组。
- ❒ 双重列表。
- ❒ Python 字典。

测试题

1. 向列表增加元素有哪些方法？
2. 从列表删除元素有哪些方法？
3. 要得到一个列表的有序副本，但又不能改变原来的列表，有哪两种方法？
4. 怎样得出某个值是否在列表中？
5. 如何确定某个值在列表中的位置？
6. 什么是元组？
7. 如何建立双重列表？
8. 如何从一个双重列表中得到一个值？
9. 什么是字典？
10. 如何向字典中增加项？
11. 怎样使用键去查找一个条目？

动手试一试

1. 写一个程序，让用户提供 5 个名字。程序要把这 5 个名字保存在一个列表中，最后打印出来。就像这样：

```
Enter 5 names:
Tony
Paul
Nick
Michel
Kevin
The names are Tony Paul Nick Michel Kevin
```

2. 修改第 1 题的程序，要求不仅显示原来的名字列表，还要显示出排序后的列表。
3. 修改第 1 题的程序，要求只显示用户键入的第 3 个名字，就像这样：

```
The third name you entered is: Nick
```

4. 修改第 1 题的程序，让用户替换其中一个名字。用户应该能选择要替换哪个名字，然后键入新名字。最后显示这个新的列表：

```
Enter 5 names:
Tony
Paul
Nick
Michel
Kevin
The names are Tony Paul Nick Michel Kevin
Replace one name. Which one? (1-5): 4
New name: Peter
The names are Tony Paul Nick Peter Kevin
```

5. 编写一个字典程序，让用户可以添加单词和定义，然后可以查找这些单词。确保当要查找的单词不存在时，用户能够知晓。运行的时候，它应该是像这样的：

```
Add or look up a word (a/l)? a
Type the word: computer
Type the definition: A machine that does very fast math
Word added!
Add or look up a word (a/l)? l
Type the word: computer
A machine that does very fast math
Add or look up a word (a/l)? l
Type the word: qwerty
That word isn’t in the dictionary yet.
```

第 13 章

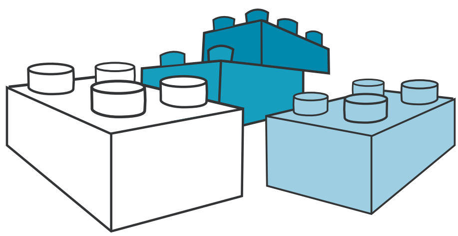
函　　　数

我们的程序很快就会变得越来越大，越来越复杂。需要一些方法把它们分成较小的部分进行组织，这样更易于编写，也更容易明白。

要把程序分解成较小的部分，主要有 3 种方法。函数（function）就像是代码的积木，可以反复地使用。利用对象（object），可以把程序中的各部分描述为自包含的单元。模块（module）就是包含程序各部分的单独的文件。在这一章中，我们将学习函数，后面两章会讨论对象和模块。学习完这些知识，我们就具备了所需要的全部基本工具，可以开始使用图形和声音并且创建游戏了。

13.1　函数——积木

最简单地讲，函数就是可以完成某个工作的代码块。这是可以用来构建更大程序的一个小部分。可以把这个小部分与其他部分放在一起，就像用积木搭房子一样。



创建或定义函数要使用 Python 的 `def` 关键字。然后可以利用函数名来使用或调用这个函数。下面先来看一个简单的例子。

创建一个函数

代码清单 13-1 中的代码首先定义了一个函数，然后使用这个函数。这个函数会在屏幕上打印一个邮件地址。

代码清单 13-1 创建和使用函数

```
def printMyAddress():
    print "Warren Sande"
    print "123 Main Street"
    print "Ottawa, Ontario, Canada"
    print "K2M 2E9"
    print

printMyAddress()
```

定义（创建）函数

调用（使用）函数

第 1 行中，我们使用 def 关键字定义了一个函数。在函数名后面有一对括号“()”，然后是一个冒号：

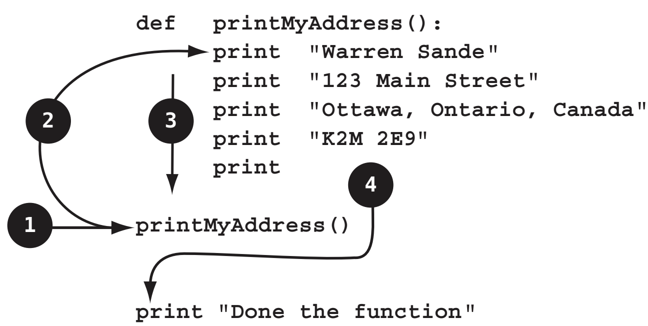
```
def printMyAddress():
```

后面很快就会解释这个括号做什么用。冒号告诉 Python 接下来是一个代码块（就像 for 循环、while 循环和 if 语句中一样）。

下面就是构成这个函数的代码。



代码清单 13-1 的最后一行是主程序：这里给出函数名和括号来调用这个函数。程序就从这里开始运行。正是这一行让程序开始运行刚才定义的函数中的代码。



主程序调用函数时，就像是这个函数在帮助主程序完成它的任务。

def 块中的代码并不是主程序的一部分，所以程序运行时，它会跳过这一部分，从 def 块以外的第一行代码开始运行。右图显示了调用函数时会发生什么。我在程序最后额外增加了一行代码，它会在函数完成后打印一条消息。

这个图中包括以下步骤。

(1) 从这里开始。这是主程序的开始。

(2) 调用函数时，跳到函数中的第一行代码。

(3) 执行函数中的每一行代码。

(4) 函数完成时，从离开主程序的那个位置继续执行。

13.2 调用函数

调用函数是指运行函数里的代码。如果我们定义了一个函数，但是从来不调用它，这些代码就永远也不会运行。

调用函数时要使用函数名和一对括号。有时括号里还会有些东西，有时也可能什么也没有。

试着运行代码清单 13-1 中的程序，看看会发生什么。你会看到这样的结果：

```
>>> =================== RESTART ===================
>>>
Warren Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9
>>>
```

从下面这个更简单的程序也可以得到同样的输出：

```
print "Warren Sande"
print "123 Main Street"
print "Ottawa, Ontario, Canada"
print "K2M 2E9"
print
```

那为什么要自找麻烦使用代码清单 13-1 中的函数让问题更复杂呢？

使用函数的主要原因是，一旦定义了函数，就可以通过调用反复地使用。所以如果我们想把地址打印 5 次，可以这样做：

```
printMyAddress()
printMyAddress()
printMyAddress()
printMyAddress()
printMyAddress()
```

输出将是：

```
Warren Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9

Warren Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9
```

```
Warren Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9

Warren Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9

Warren Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9
```

你可能会说：可以不用函数，用循环也能做同样的事情。



我就知道你会这么讲……对于这种情况，你确实可以用循环做同样的事情。不过，如果希望在程序的不同位置打印地址，而不是全部都一次完成，循环就实现不了了。

使用函数还有一个原因，每次函数运行时可以让它有不同的表现。我们将在下一节了解这是如何做到的。

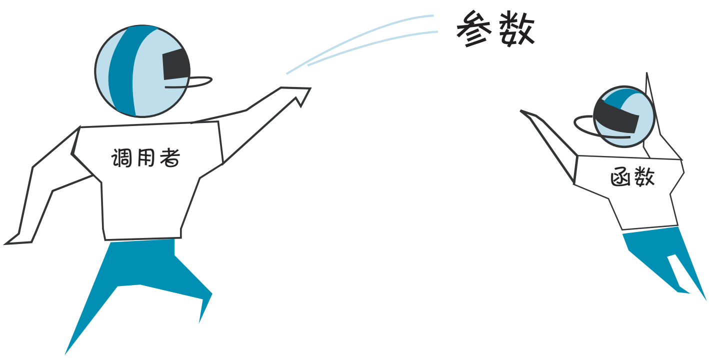
13.3 向函数传递参数

现在来看括号做什么用：它用来传递参数（argument）！



不，卡特，计算机非常听话，它们永远也不会争论[①]。在编程中，参数这个词是指你交给函数的一条信息。我们把这称为：你向函数传递参数。

① argument 也有“争论”的意思，卡特显然是把这里的 argument 理解为“争论”了。——编者注



假设你希望对你的所有家庭成员使用这个地址打印函数。所有人的地址都是一样的，但是每一次人名会有所不同。不能在函数中把人名硬编码写成 Warren Sande，你可以建立一个变量。调用函数时将这个变量传递到函数。

要说明这是如何工作的，最容易的方法就是举例子。在代码清单 13-2 中，我修改了地址打印函数，要使用一个对应人名的参数。参数是有名字的，就像其他变量一样。我把这个变量命名为 myName。

函数运行时，变量 myName 会填入调用函数时为它传入的任何参数。调用函数时，我们把参数放在括号里，通过这种方式将参数传入函数。

因此，在代码清单 13-2 中，参数 myName 赋值为 Carter Sande。

代码清单 13-2　向函数传递参数

```
def printMyAddress(myName):        ← 将 myName 参数传入函数
    print myName       ← 打印人名
    print "123 Main Street"
    print "Ottawa, Ontario", Canada
    print "K2M 2E9"
    print

printMyAddress("Carter Sande")
```

将“卡特 Sande”作为参数传入函数；函数中的变量 myName 的值将是“卡特 Sande”

运行代码，你会得到期望的结果：

```
>>> ===================== RESTART =====================
>>>
Carter Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9
```

这看上去与第一个程序（没有使用参数）得到的输出完全相同。不过，我们每次可以用不同方式打印地址，比如：

```
printMyAddress("Carter Sande")
printMyAddress("Warren Sande")
printMyAddress("Kyra Sande")
printMyAddress("Patricia Sande")
```

现在每次调用函数时输出都不同。人名会变，因为我们每次都向函数传入了不同的人名。

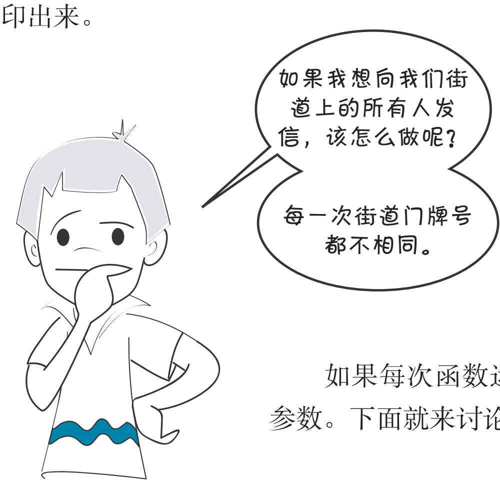
```
>>> ============================ RESTART ============================
>>>
Carter Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9

Warren Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9

Kyra Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9

Patricia Sande
123 Main Street
Ottawa, Ontario, Canada
K2M 2E9
```

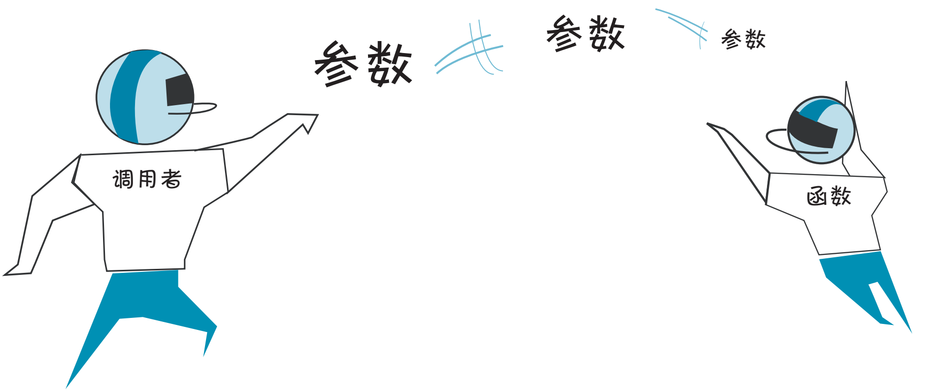
注意，我们向函数传入什么值，函数中就会使用什么值，并作为地址的人名部分打印出来。



如果每次函数运行时有多个信息不同，就需要多个参数。下面就来讨论这个问题。

13.4 有多个参数的函数

在代码清单 13-2 中，我们的函数只有一个参数。不过函数完全可以有多个参数。实际上，你想要有多少个参数就可以有多少个参数。下面来看一个带两个参数的例子，我想，通过这个例子，你会对多个参数有所认识。在这个基础上，你可以根据具体需要为程序中的函数增加参数。



术语箱

谈到向函数传递信息时，你可能还会听到这样一个词：形参（parameter）。有些人说参数（argument）和形参（parameter）可以互换。所以你可以说，

“我向这个函数传递两个形参（parameter）”，或者

“我向这个函数传递两个参数（argument）”。

不过有些人认为，谈到传递部分（调用函数）时应当称作实参（argument），而谈到接收部分（函数内部）时应该称为形参（parameter）。



使用参数（不论是 argument 还是 parameter）讨论向函数传递值时，程序员都明白你是什么意思。

要向街道上的每一个人发送卡特的信，我们的地址打印函数需要两个参数：一个对应人名，另一个对应门牌号码。代码清单 13-3 显示了这个函数。

代码清单 13-3　带两个参数的函数

```
def printMyAddress(someName, houseNum):
    print someName
    print houseNum,
    print "Main Street"
    print "Ottawa, Ontario, Canada"
    print "K2M 2E9"
    print

printMyAddress("Carter Sande", "45")
printMyAddress("Jack Black", "64")
printMyAddress("Tom Green", "22")
printMyAddress("Todd White", "36")
```

使用两个变量，对应两个参数

两个变量都要打印

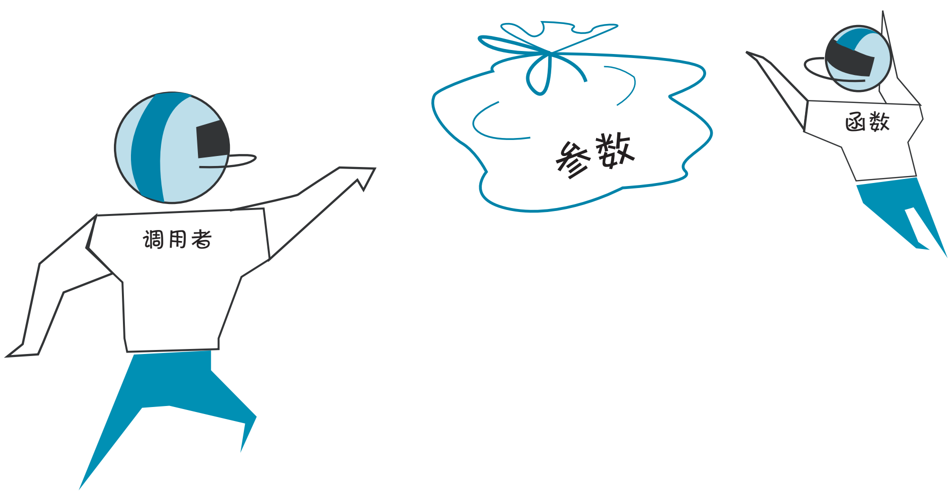
逗号使门牌号和街道显示在同一行上

调用函数并传入两个参数

使用多个参数时，要用逗号来分隔，就像列表中的元素一样，这就引入了下一个话题……

多少才算太多

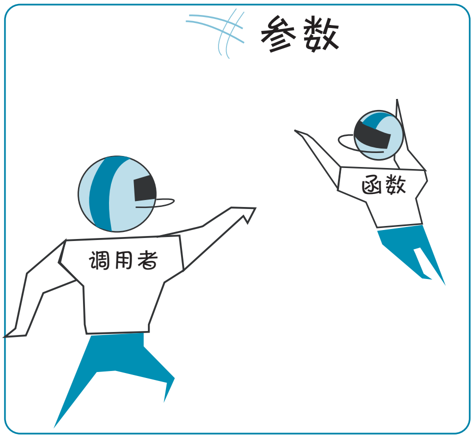
前面说过，想向函数传递多少参数就可以有多少个参数。这一点不假，但是如果你的函数有超过 5 到 6 个参数，可能就应该考虑采用别的做法了。一种做法是把所有参数收集到一个列表中，然后把这个列表传递到函数。这样一来，就只是传递一个变量（列表变量），只不过其中包含有一组值。这样可以让你的代码更易读。

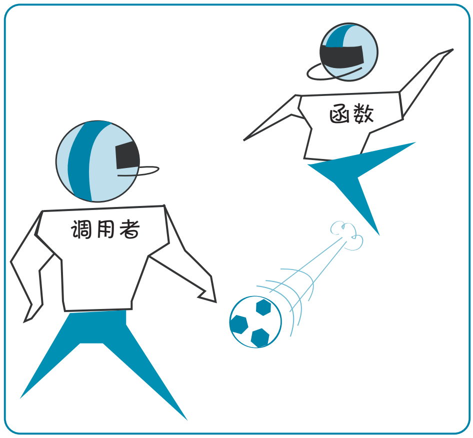


13.5　返回值的函数

目前为止，函数只是为我们做一些工作。不过函数的一个突出作用是：它们还可以向你发回一些东西。

我们已经知道，可以向函数发送信息（参数），不过函数还可以向调用者发回信息。从函数返回的值称为结果（result）或返回值（return value）。





返回一个值

要让函数返回一个值，需要在函数中使用 Python 关键字 return。下面给出一个例子：

```
def calculateTax(price, tax_rate):
    taxTotal = price + (price * tax_rate)
    return taxTotal
```

这会把值 taxTotal 发回到调用这个函数的程序部分。

不过发回这个值时，它会去哪里呢？返回值会回到调用这个函数的代码。看下面的例子：

```
totalPrice = calculateTax(7.99, 0.06)
```

calculateTax 函数会返回一个值：8.4694，这个值将赋给 totalPrice。

使用表达式的任何地方都可以使用函数来返回值。可以把返回值赋给一个变量（就像前面一样），也可以在另一个表达式中使用，或者打印出来，例如：

```
>>>print calculateTax(7.99, 0.06)
8.4694
>>>total = calculateTax(7.99, 0.06) + calculateTax(6.59, 0.08)
```

对返回值也可以不做任何处理，就像这样：

```
>>>calculateTax(7.49, 0.07)
```

在上面这个例子中，函数会运行，计算出税后总价格，不过我们没有使用这个结果。

下面用一个有返回值的函数建立程序。在代码清单 13-4 中，calculateTax() 函数返回了一个值。向这个函数提供税前价格和税率，它会返回税后价格。我们把这个值赋给一个变量。所以不像前面那样只是使用函数的名，这里还需要一个变量和一个等号（=），然后是函数名。变量会赋为 calculateTax() 函数返回的结果。

代码清单 13-4　创建和使用有返回值的函数

```
def calculateTax(price, tax_rate):
    total = price + (price * tax_rate)
    return total

my_price = float(raw_input ("Enter a price: "))

totalPrice = calculateTax(my_price, 0.06)
print "price = ", my_price, " Total price = ", totalPrice
```

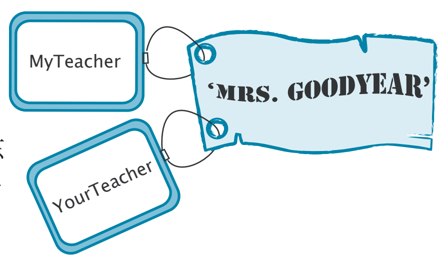
函数计算税额，并返回总价格

将结果发回给主程序

调用函数并把结果保存在 totalPrice

试着键入代码清单 13-4 中的程序，保存并运行这个程序。注意这个代码中的税率固定为 0.06（等于 6 个百分点）。如果程序必须处理不同的税率，可以让用户输入价格的同时还要输入税率。

13.6　变量作用域



你可能已经注意到，有些变量在函数之外，如 totalPrice，还有一些变量在函数内部，如 total。这些变量只是同一个东西的两个不同名字。这就像第 2 章中所说的 YourTeacher = MyTeacher。

在我们的 calculateTax 例子中，totalPrice 和 total 是贴在同一个东西上的两个标签。对于函数而言，函数内的名字只是在函数运行时才会创建。在函数运行之前或者完成运行之后甚至根本不存在。Python 提供了内存管理（memory management），可以自动完成这个工作。Python 在函数运行时会创建新的名字在函数内使用，当函数完成时会把它们删除。最后这部分很重要：函数运行结束时，其中的所有名字都不再存在。

函数运行时，函数之外的名字被搁置一边，而没有用到。只有函数内部的名字会被用到。程序中使用（或者可以使用）变量的部分称为这个变量的作用域（scope）。

局部变量

在代码清单 13-4 中，变量 price 和 total 只在函数内使用。我们说 price、total 和 tax_rate 的作用域是 calculateTax() 函数。这也称为这些变量是局部的（local）。price、total 和 tax_rate 变量是 calculateTax() 函数中的局部变量。

要了解这是什么意思，一种方法是向代码清单 13-4 中的程序增加一行代码，尝试在函数之外的某个位置打印 price 的值。代码清单 13-5 做了这个尝试。

代码清单 13-5　尝试打印一个局部变量

```
def calculateTax(price, tax_rate):          定义一个函数计算税额
    total = price + (price * tax_rate)      并返回总价格
    return total

my_price = float(raw_input ("Enter a price: "))

totalPrice = calculateTax(my_price, 0.06)   <-- 调用函数，保存并打印结果
print "price = ", my_price, " Total price = ", totalPrice
print price   <-- 尝试打印 price
```

如果运行这个程序，会得到这样一个错误：

```
Traceback (most recent call last):
  File "C:/.../Listing_13-5.py", line 9 , in <module>
    print price
NameError: name 'price' is not defined   <-- 这一行解释了错误
```

错误消息的最后一行解释了这个问题的原委：在 calculateTax() 函数以外，变量 price 根本没有定义。它只是在函数运行时才存在。试图在这个函数之外打印 price 的值时（此时函数并没有运行），就会得到一个错误。

全局变量

与局部变量 price 对应，代码清单 13-5 中的变量 my_price 和 totalPrice 在函数之外定义（程序主部分中）。我们使用全局变量（global）表示有更大作用域的变量。在这种情况下，更大是指程序的主部分，而不是函数内部。如果扩展代码清单 13-5 中的程序，完全可以在另一个位置使用变量 my_price 和 totalPrice，它们仍然有之前给定的值。它们仍在合法的作用域中（in scope）。因为我们可以在程序的任何地方使用这些变量，所以把它们称作全局变量（global variable）。

在代码清单 13-5 中，试图在函数之外打印一个函数内的变量时，会得到一条错误消息。这个变量不存在，也就是说它在作用域之外（out of scope）。如果反过来：从函数内打印一个全局变量，你认为会发生什么？

代码清单 13-6 试图从 calculateTax() 函数中打印变量 my_price。试试看会发生什么。

代码清单 13-6 在函数中使用全局变量

```
def calculateTax(price, tax_rate):
    total = price + (price * tax_rate)
    print my_price          ← 尝试打印 my_price
    return total

my_price = float(raw_input ("Enter a price: "))

totalPrice = calculateTax(my_price, 0.06)
print "price = ", my_price, " Total price = ", totalPrice
```

可以吗？真的可以！不过为什么呢？

开始讨论变量作用域时，我曾经说过，Python 利用内存管理在函数运行时自动创建局部变量。内存管理还会做其他事情。如果在函数中使用主程序中定义的变量名，Python 允许你使用这个全局变量，只要你不要试图改变它。

所以你可以这样做：

```
print my_price
```

或者这样做：

```
your_price = my_price
```

因为它们都不会改变 my_price。

如果函数的任何部分试图改变这个变量，Python 会创建一个新的局部变量。所以如果你打算这样做：

```
my_price = my_price + 10
```

那么 my_price 将是 Python 在函数运行时创建的一个新的局部变量。

在代码清单 13-6 的例子中，打印出的值是全局变量 my_price，因为函数没有改变这个变量。代码清单 13-7 中的程序表明，如果确实试图在函数内部改变全局变量，你会得到一个新的局部变量。试着运行这个程序，看看会有什么结果。

代码清单 13-7　尝试在函数内部修改一个全局变量

```
def calculateTax(price, tax_rate):
    total = price + (price * tax_rate)

    my_price = 10000
    print "my_price (inside function) = ", my_price
    return total

my_price = float(raw_input ("Enter a price: "))

totalPrice = calculateTax(my_price, 0.06)
print "price = ", my_price, " Total price = ", totalPrice
print "my_price (outside function) = ", my_price
```

在函数内部修改 my_price

打印局部版本的 my_price

这里的变量 my_price 与这里的 my_price 是完全不同的内存块

打印全局版本的 my_price

如果运行代码清单 13-7 中的代码，会有下面的输出：

```
>>> ========================== RESTART ==========================
>>>
Enter a price: 7.99
my_price (inside function) =  10000
price =  7.99  Total price =  8.4694
my_price (outside function) =  7.99
>>>
```

从函数内打印 my_price

从函数外打印 my_price

可以看到，现在有两个名为 my_price 的不同变量，分别有不同的值。一个是 calculateTax() 函数中的局部变量，我们将它设置为 10 000。另一个是主程序中定义的全局变量，用来获取用户的输入，它的值是 7.99。

13.7　强制为全局

上一节中，我们看到，如果试图从函数内改变一个全局变量的值，Python 会创建一个新的局部变量。这是为了防止函数无意地改变全局变量。

不过，有些情况下确实要在函数中改变一个全局变量。这该怎么做呢？

可以用 Python 的一个关键字 global 来做到。可以这样来使用：

```
def calculateTax(price, tax_rate):
    global my_price
```

告诉 Python 你想使用全局版本的 my_price

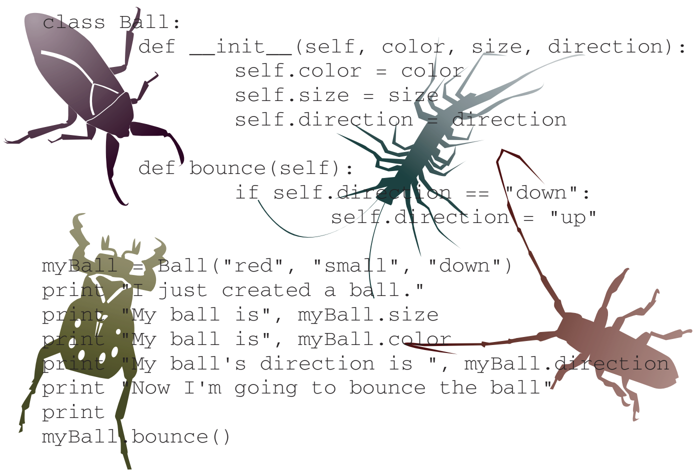
如果使用 global 关键字，Python 不会建立名为 my_price 的局部变量，而是会使用名为 my_price 的全局变量。另外，如果还没有名为 my_price 的全局变量，Python 就会创建一个。

13.8 关于变量命名的一点建议

在前面的几节中已经看到，可以对全局变量和局部变量使用相同的变量名。Python 会在需要时自动创建新的局部变量，或者也可以用 `global` 关键字阻止它创建。不过，我强烈建议你不要重复使用变量名。

你可能已经从一些例子中注意到，往往很难知道一个变量是局部的还是全局的，这让代码更加混乱，因为存在同名的不同变量。而且，只要有混乱，错误就会乘虚而入。

所以对目前的状况来说，建议你对局部变量和全局变量使用不同的名字。这样就不会有混乱，也能把错误拒之门外。



你学到了什么

在这一章，你学到了以下内容。

- ❒ 什么是函数。
- ❒ 什么是参数（argument 或 parameter）。
- ❒ 如何向函数传递一个参数。
- ❒ 如何向函数传递多个参数。
- ❒ 如何让函数向调用者返回一个值。
- ❒ 变量作用域是什么，什么是局部变量和全局变量。
- ❒ 如何在函数中使用全局变量。

测试题

1. 使用哪个关键字来创建函数？
2. 如何调用函数？
3. 如何向函数传递信息（参数）？
4. 函数最多可以有多少个参数？
5. 如何从函数返回信息？
6. 函数运行结束后，函数中的局部变量会发生什么？

动手试一试

1. 编写一个函数，用大写字母打印你的名字，就像这样：

```
  CCCC       A       RRRRR  TTTTTTT  EEEEEE  RRRRR
 C    C     A A      R    R    T     E       R    R
C          A   A     R    R    T     EEEE    R    R
C         AAAAAAA    RRRRR     T     E       RRRRR
 C    C  A       A   R    R    T     E       R    R
  CCCC  A         A  R     R   T     EEEEEE  R     R
```

   编写一个程序多次调用这个函数。

2. 建立一个函数，可以打印全世界任何人名、地址、街道、城市、州或省、邮政编码和国家。（提示：这需要 7 个参数。可以把它们作为单独的参数传入，也可以作为一个列表。）
3. 尝试使用代码清单 13-7 的例子，不过要求 my_price 是全局变量，以便看到结果输出有什么区别。
4. 编写一个函数计算零钱的总面值，包括五分币、二分币和一分币（类似于第 5 章中最后一个“动手试一试”问题）。函数应当返回这些硬币的总面值。然后编写一个程序调用这个函数。程序运行时应当得到类似下面的输出：

```
quarters: 3
dimes: 6
nickels: 7
pennies: 2
total is $1.72
```

第 14 章
对　　象

在前几章中，我们已经了解了可以使用不同方式组织数据和程序，以及把东西收集在一起。我们看到了列表可以收集变量（数据），函数可以把一些代码收集到能够反复使用的单元中。



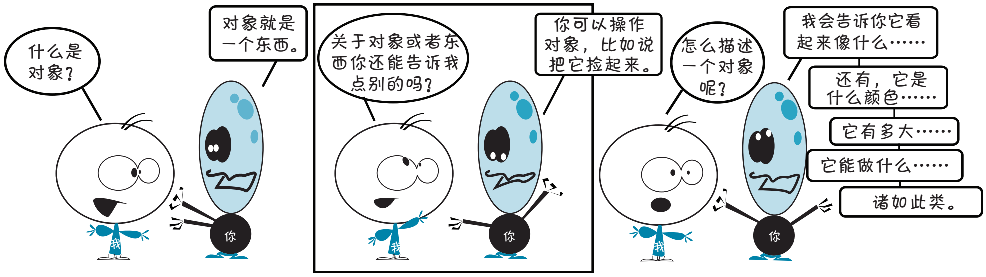
对象（object）则让这种收集的思想更向前迈进一步。对象可以把函数和数据收集在一起。这个主意在编程中非常有用，而且在很多很多的程序中都已经用到。实际上，如果仔细分析 Python，几乎一切都是对象。按编程的术语来讲，我们说 Python 是面向对象的（object oriented）。这说明，Python 中可以使用对象（实际上这也相当容易）。并不是一定得创建自己的对象，不过这样可以让很多事情更容易一些。

在这一章中，我们将学习什么是对象，以及如何创建和使用对象。后面几章开始处理图形时，我们将会大量使用对象。

14.1　真实世界中的对象

什么是对象？如果我们不是在讨论编程，当我问到这个问题时，可能会有下面的对话：

"我"与"你"的对话



在 Python 中定义什么是对象也可以作为一个很好的起点。拿球来举个例子。可以操作一个球，比如捡球、抛球、踢球或者充气（对于某些球来说）。我们把这些操作称为动作（action）。还可以通过指出球的颜色、大小和重量来描述一个球。这些就是球的属性（attribute）。

术语箱

可以通过描述特征或属性来描述一个对象。球的属性之一是它的形状。大多数球都是圆形。还有一些其他的属性，比如颜色、大小、重量和价格。属性的另一个说法是特性（property）。

真实世界的真实对象（物体）包括两个方面。

- 可以对它们做什么（动作）。
- 如何描述（属性或特性）。

编程中也是如此。

14.2　Python 中的对象

在 Python 中，一个对象的特征（或"你知道的事情"）也称为属性（attribute），这应该很好记。动作（或"能够对对象做的操作"）称为方法（method）。

如果要建立一个球的 Python 版本或者模型（model），球就是一个对象，它要有属性和方法。

球的属性可能包括：

```
ball.color
ball.size
ball.weight
```

这些都是关于球的描述。

球的方法可能包括：

```
ball.kick()
ball.throw()
ball.inflate()
```

这些都是可以对球做的操作。

什么是属性

属性就是你所知道（或者可以得出）的关于球的所有方面。球的属性就是一些信息（数字、字符串等等）。听起来很熟悉？没错，它们就是变量，只不过是包含在对象中的变量。

可以显示：

```
print ball.size
```

可以为它们赋值：

```
ball.color = 'green'
```

可以把它们赋给常规的、不是对象的变量：

```
myColor = ball.color
```

还可以把它们赋给其他对象的属性：

```
myBall.color = yourBall.color
```

什么是方法

方法就是可以对对象做的操作，它们是一些代码块，可以调用这些代码块来完成某个工作。听起来很熟悉？没错，方法就是包含在对象中的函数。

函数能做到的，方法都可以做到，包括传递参数和返回值。

14.3 对象 = 属性 + 方法

所以利用对象，可以把一个东西的属性和方法（你知道的事情和你可以做的事情）收集在一起。属性是信息，方法是动作。

14.4 这个点是什么

在前面的球例子中，你可能已经注意到对象名与属性或方法名之间的点。这是 Python 使用对象属性和方法的一种记法：`object.attribute` 或 `object.method()`。就这么简单。这称为点记法，很多编程语言中都使用了这种记法。

现在对于对象已经有了整体认识。下面来建立一些对象！

14.5　创建对象



Python 中创建对象包括两步。

第一步是定义对象看上去什么样，会做什么，也就是它的属性和方法。但是创建这个描述并不会真正创建一个对象。这有点像一个房子的蓝图。蓝图可以告诉你房子看上去怎么样，但是蓝图本身并不是一个房子。你不可能住在一个蓝图里。只能用它来建造真正的房子。实际上，可以使用蓝图盖很多的房子。

在 Python 中，对象的描述或蓝图称为一个类（class）。

第二步是使用类来建立一个真正的对象。这个对象称为这个类的一个实例（instance）。

下面来看一个建立类和实例的例子。代码清单 14-1 显示了一个简单的 `Ball` 类的类定义。

代码清单 14-1　创建一个简单的 Ball 类

```
class Ball:          ← 这里告诉 Python
                       我们在建立一个类

    def bounce(self):
        if self.direction == "down":        这是一个方法
            self.direction = "up"
```

代码清单 14-1 是一个球的类定义，其中只有一个方法 `bounce()`。不过，属性呢？嗯，属性并不属于类，它们属于各个实例。因为每个实例可以有不同的属性。

设置实例属性有两种方法。后面的小节中我们会分别了解这两种方法。

创建一个对象实例

前面提到过，类定义并不是一个对象。这只是蓝图。现在来盖真正的房子。

如果想创建 Ball 的一个实例，可以这样做：

```
myBall = Ball()
```

这个球还没有任何属性，所以下面给它提供一些属性：

```
myBall.direction = "down"
myBall.color = "green"
myBall.size = "small"
```

这是为对象定义属性的一种方法。下一节还会学习另一种方法。

现在来试试它的方法。我们要这样使用 bounce() 方法：

```
myBall.bounce()
```

下面把这些都放在一个程序里，增加一些 print 语句来看发生了什么。程序见代码清单 14-2。

代码清单 14-2　使用 Ball 类

```
class Ball:

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"

myBall = Ball()
myBall.direction = "down"
myBall.color = "red"
myBall.size = "small"

print "I just created a ball."
print "My ball is", myBall.size
print "My ball is", myBall.color
print "My ball's direction is", myBall.direction
print "Now I'm going to bounce the ball"
print
myBall.bounce()
print "Now the ball's direction is", myBall.direction
```

这就是我们的类，与前面相同

建立类的一个实例

设置一些属性

打印对象的属性

使用一个方法

运行这个程序，可以看到下面的结果：

```
>>> ========================== RESTART ==========================
>>>
I just created a ball.
My ball is small
My ball is red
My ball's direction is down
Now I'm going to bounce the ball

Now the ball's direction is up
```

我们设置的属性

现在调用 bounce() 让球反弹

它会改变方向，从下（down）改为上（up）

注意，调用 bounce() 方法会把球的方向（direction）从下（down）改为上（up），这正是 bounce() 方法中的代码所要做的。

初始化对象

创建球对象时，并没有在 size、color 或 direction 中填入任何内容。必须在创建对象之后填充这些内容。不过有一种方法可以在创建对象时设置属性。这称为初始化对象。

术语箱

初始化（initializing）表示“开始时做好准备”。在软件中对某个东西初始化时，就是把它设置成一种我们希望的状态或条件，以备使用。

创建类定义时，可以定义一个特定的方法，名为 __init__()，只要创建这个类的一个新实例，就会运行这个方法。可以向 __init__() 方法传递参数，这样创建实例时就会把属性设置为你希望的值。代码清单 14-3 显示了这是如何实现的。

代码清单 14-3　增加一个 __init__() 方法

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"

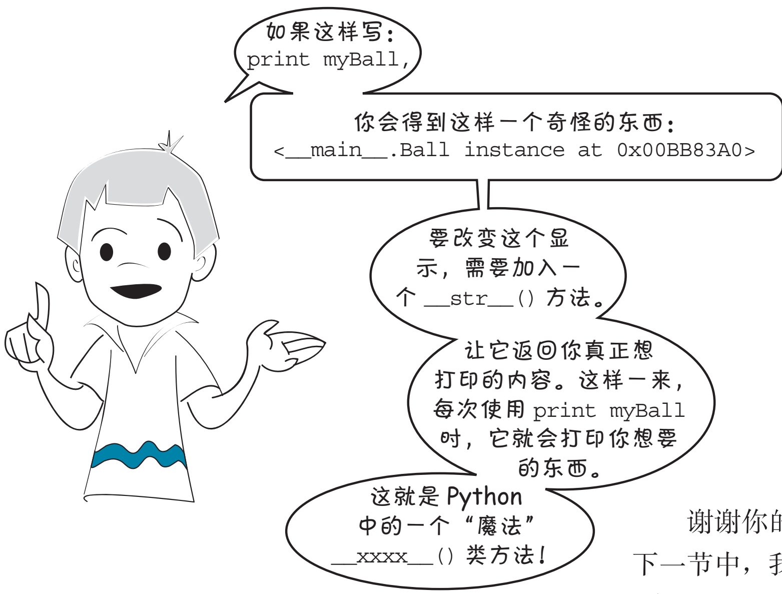
myBall = Ball("red", "small", "down")
print "I just created a ball."

print "My ball is", myBall.size
print "My ball is", myBall.color
print "My ball's direction is ", myBall.direction
print "Now I'm going to bounce the ball"
print
myBall.bounce()
print "Now the ball's direction is", myBall.direction
```

这里是 __init__() 方法。init 前后各有两条下划线，总共四条下划线

属性作为 __init__() 的参数传入

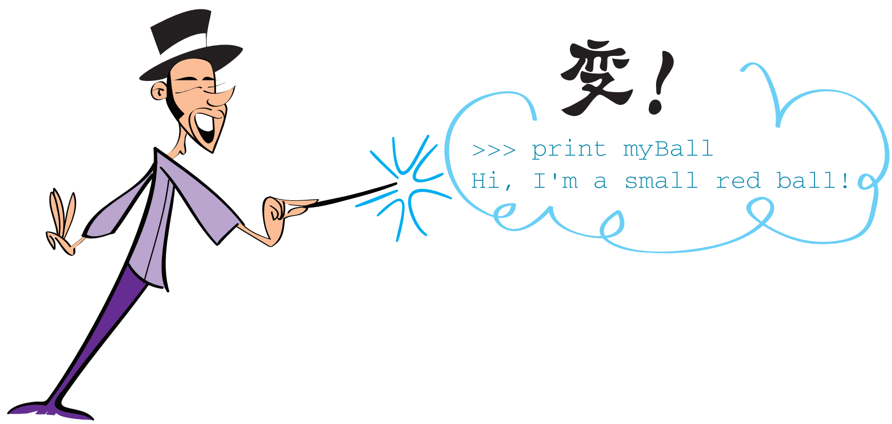
如果这个程序，得到的输出应该与代码清单 14-2 的相同。区别在于，代码清单 14-3 使用了 __init__() 方法来设置属性。



谢谢你的提醒，卡特。在下一节中，我们将会了解这些“魔法”方法到底是什么。

“魔法”方法：__str__()

就像卡特说的，Python 中的对象有一些“魔法”方法，当然它们并不是真的有魔法！这些只是在你创建类时 Python 自动包含的一些方法。Python 程序员通常把它们叫做特殊方法（special method）。



我们已经知道，__init__() 方法会在对象创建时完成初始化。每个对象都内置有一个 __init__() 方法。如果你在类定义中没有加入自己的 __init__() 方法，就会有这样一个内置方法接管，它的工作就是创建对象。

另一个特殊方法是 __str__()，它会告诉 Python 打印（print）一个对象时具体显示什么内容。Python 会默认以下内容。

- 实例在哪里定义（卡特的例子中，就是在 __main__ 中，这是程序的主部分）。
- 类名（Ball）。
- 存储实例的内存位置（0x00BB83A0 部分）。

不过，如果你希望 print 为对象显示其他的内容，可以定义自己的 __str__()，这会覆盖内置的 __str__() 方法。代码清单 14-4 举了个例子。

代码清单 14-4　使用 __str__() 改变打印对象的方式

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction

    def __str__(self):
        msg = "Hi, I'm a " + self.size + " " + self.color + " ball!"
        return msg
myBall = Ball("red", "small", "down")
print myBall
```

这里是 __str__() 方法

现在运行这个程序，可以得到下面的结果：

```
>>> ================= RESTART =================
>>>
Hi, I'm a small red ball!
```

这看起来比 <__main__.Ball instance at 0x00BB83A0> 好多了，你认为呢？所有“魔法”方法都在方法名称前后各使用两条下划线。

什么是 self

你可能已经注意到，在类属性和方法定义中多处出现了“self”，比如：

```
def bounce(self):
```

self 是什么意思？嗯，我们说过，可以使用蓝图盖很多个房子，还记得吧？使用一个类也可以创建多个对象实例，例如：

```
cartersBall = Ball("red", "small", "down")
warrensBall = Ball("green", "medium", "up")
```

创建 Ball 类的两个实例

调用其中一个实例的方法时，像这样：

```
warrensBall.bounce()
```

方法必须知道是哪个实例调用了它。是 cartersBall 需要反弹吗？还是 warrensBall ？self 参数会告诉方法哪个对象调用它。这称为实例引用（instance reference）。

不过先等等！调用方法时，warrensBall.bounce() 的括号里没有参数，但是方法里却有一个 self 参数。既然我们并没有传入任何东西，这个 self 参数从哪里来的？这是 Python 处理对象的另外一个“魔法”。调用一个类方法时，究竟是哪个实例调用了这个方法？这个信息（也就是实例引用）会自动传递给方法。

这就像写成：

```
Ball.bounce(warrensBall)
```

在这种情况下，我们告诉了 bounce() 方法哪个球要反弹。实际上，这个代码也能正常工作，因为写成 warrensBall.bounce() 时，Python 在后台确实也是这么做的。

> 顺便说一句，self 这个名字在 Python 中没有任何特殊的含义。只不过所有人都使用这个实例引用名。这也是让代码更易读的一个约定。也可以把这个实例变量命名为你想要的任何名字，不过强烈建议你遵循这个约定，因为使用 self 能减少混乱。

我们在第 11 章建立了一个热狗程序。现在作为使用对象的例子，我们来为热狗建立一个类。

14.6 一个示例类—— HotDog

在这个例子中，我们假设热狗总包括一个小面包。（否则可真是一团糟。）下面为热狗指定一些属性和方法。

下面是热狗的属性。

- cooked_level：这是一个数字，通过这个属性我们可以知道热狗烤了多长时间。0 ～ 3 表示还是生的，超过 3 表示半生不熟，超过 5 表示已经烤好，超过 8 表示已经烤成木炭了！我们的热狗开始时是生的。
- cooked_string：这是一个字符串，描述热狗的生熟程度。
- condiments：这是热狗上的配料列表，比如番茄酱、芥末酱等。

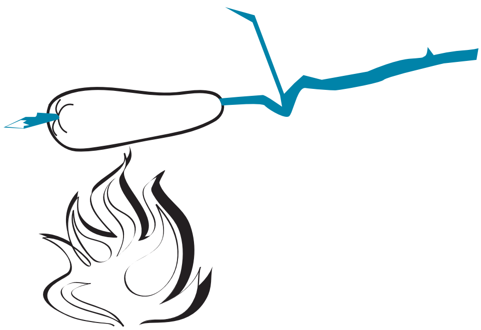
下面是热狗的方法。

- ❒ cook()：把热狗烤一段时间。这会让热狗越来越熟。
- ❒ add_condiment()：给热狗加一些配料。
- ❒ __init__()：创建实例并设置默认属性。
- ❒ __str__()：让 print 的结果看起来更好一些。

首先，需要定义类。先定义 __init__() 方法，它会为热狗设置默认属性：

```
class HotDog:
    def __init__(self):
        self.cooked_level = 0
        self.cooked_string = "Raw"
        self.condiments = []
```

先从一个没有加任何配料的生热狗开始。



现在，来建立一个方法烤热狗：

```
def cook(self, time):
        self.cooked_level = self.cooked_level + time   ← 按 time（时间）量增加烤制级别
        if self.cooked_level > 8:
            self.cooked_string = "Charcoal"
        elif self.cooked_level > 5:
            self.cooked_string = "Well-done"          为不同烤制级别设置字符串
        elif self.cooked_level > 3:
            self.cooked_string = "Medium"
        else:
            self.cooked_string = "Raw"
```

继续下面的工作之前，先对这一部分做个测试。首先，需要创建热狗的一个实例，还要检查它的属性。

```
myDog = HotDog()
print myDog.cooked_level
print myDog.cooked_string
print myDog.condiments
```

下面把这些内容都放在一个程序中，运行这个程序。代码清单 14-5 显示了（到目前为止）完整的程序。

代码清单 14-5 热狗程序的开始部分

```
class HotDog:
    def __init__(self):
        self.cooked_level = 0
        self.cooked_string = "Raw"
        self.condiments = []
    def cook(self, time):
        self.cooked_level = self.cooked_level + time
        if self.cooked_level > 8:
            self.cooked_string = "Charcoal"
        elif self.cooked_level > 5:
            self.cooked_string = "Well-done"
        elif self.cooked_level > 3:
            self.cooked_string = "Medium"
        else:
            self.cooked_string = "Raw"
myDog = HotDog()
print myDog.cooked_level
print myDog.cooked_string
print myDog.condiments
```

像（Python）程序员一样思考

Python 中的另一个约定是类名总是以大写字母开头。目前为止，我们已经见到 Ball 和 HotDog，所以说我们一直都在遵循这个约定。



现在，运行代码清单 14-5 中的代码，看看会得到什么。结果应该像这样：

```
>>>
0        <- The cooked_level
Raw      <- The cooked_string
[]       <- The condiments
```

可以看到，属性分别是 cooked_level = 0，cooked_string = "Raw"，另外 condiments 为空。

现在来测试 cook() 方法。把下面的代码行增加到代码清单 14-5 中：

```
print "Now I'm going to cook the hot dog"
myDog.cook(4)                          # 把热狗烤 4 分钟
print myDog.cooked_level
print myDog.cooked_string              # 检查新的 cooked 属性
```

再运行这个程序，现在输出会变成：

```
>>>
0
Raw          烤前
[]
Now I'm going to cook the hot dog
4
Medium       烤后
```

看来我们的 cook() 方法能正常工作。cooked_level 从 0 变成 4，而且字符串也得到更新（从 Raw 变成 Medium）。

下面来增加一些配料。这需要一个新的方法。另外还可以自己增加 __str__() 函数，让打印对象更为容易。按代码清单 14-6 编辑程序。

代码清单 14-6 包含 cook()、add_condiments() 和 __str__() 的 HotDog 类

```
class HotDog:
    def __init__(self):
        self.cooked_level = 0
        self.cooked_string = "Raw"
        self.condiments = []
    def __str__(self):
        msg = "hot dog"
        if len(self.condiments) > 0:
            msg = msg + " with "
        for i in self.condiments:
            msg = msg+i+", "
        msg = msg.strip(", ")
        msg = self.cooked_string + " " + msg + "."
        return msg
    def cook(self, time):
        self.cooked_level=self.cooked_level+time
        if self.cooked_level > 8:
            self.cooked_string = "Charcoal"
        elif self.cooked_level > 5:
            self.cooked_string = "Well-done"
        elif self.cooked_level > 3:
            self.cooked_string = "Medium"
        else:
            self.cooked_string = "Raw"
    def addCondiment(self, condiment):
         self.condiments.append(condiment)
```

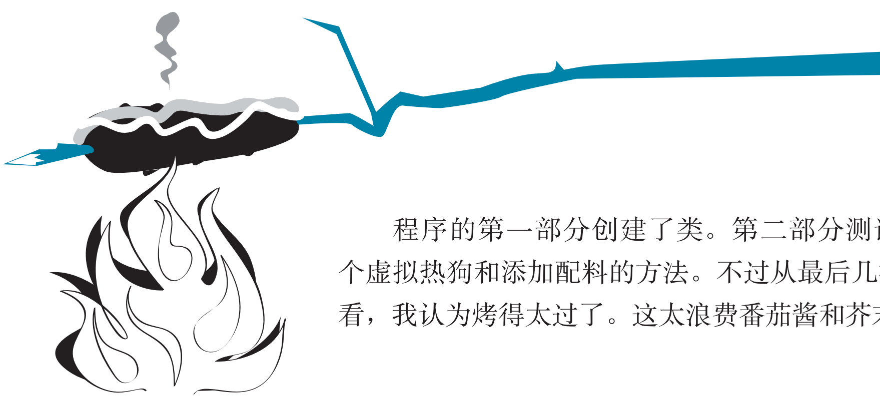
（注释：__str__() 部分——定义新的 __str__() 方法；addCondiment 部分——定义新的 add_condiments() 方法；整个代码——定义类）

```
myDog = HotDog()          ◁———— 创建实例
print myDog
print "Cooking hot dog for 4 minutes..."
myDog.cook(4)
print myDog
print "Cooking hot dog for 3 more minutes..."
myDog.cook(3)
print myDog
print "What happens if I cook it for 10 more minutes?"    查看是否一切正常
myDog.cook(10)
print myDog
print "Now, I'm going to add some stuff on my hot dog"
myDog.addCondiment("ketchup")
myDog.addCondiment("mustard")
print myDog
```

这个代码清单有点儿长，但我还是建议你自己键入这些代码，而且你已经有了之前代码清单 14-5 中的部分代码，不过，如果你的手指确实很累，或者你没有时间，也可以在 \examples 文件夹或本书网站上找到这个代码。

运行这个程序，看看能得到什么。结果应该如下：

```
>>> ================================ RESTART ============================
>>>
Raw hot dog.
Cooking hot dog for 4 minutes...
Medium hot dog.
Cooking hot dog for 3 more minutes...
Well-done hot dog.
What happens if I cook it for 10 more minutes?
Charcoal hot dog.
Now, I'm going to add some stuff on my hot dog
Charcoal hot dog with ketchup, mustard.
```



程序的第一部分创建了类。第二部分测试了烤这个虚拟热狗和添加配料的方法。不过从最后几行代码来看，我认为烤得太过了。这太浪费番茄酱和芥末酱了！

14.7　隐藏数据

你可能已经意识到，查看或修改对象中的数据（属性）有两种方法。可以直接访问，像这样：

```
myDog.cooked_level = 5
```

或者也可以使用修改属性的方法，例如：

```
myDog.cook(5)
```

如果热狗开始时是生的（cooked_level = 0），这两种做法的作用相同。它们都会把 cooked_level 设置为 5。那么为什么还要那么麻烦，专门建立一个方法来做这个工作呢？为什么不直接修改呢？

我可以想到至少两个原因。

- 如果直接访问属性，烤热狗至少需要两部分：改变 cooked_level 和改变 cooked_string。而利用一个方法，可以只做一个方法调用，它就会完成我们需要的一切工作。
- 如果直接访问属性，就会有这样的结果：

```
cooked_level = cooked_level - 2
```

这会使热狗比以前还生。不过热狗肯定不会越烤越生！所以这是毫无意义的。通过使用方法，可以确保 cooked_level 只会增加而不会减少。

术语箱

> 按编程术语来讲，如果限制对对象数据的访问，使得只能通过使用方法来获取和修改这些数据，就称为数据隐藏（data hiding）。Python 没有提供任何途径来保证数据隐藏，不过如果你愿意，可以适当地编写代码来遵循这个规则。

目前为止，我们已经看到对象包含属性和方法。而且了解了如何创建对象以及如何利用一个名为 __init__() 的特殊方法初始化对象。我们还看到了另一个特殊方法 __str__()，利用这个方法可以更好地打印我们的对象。

14.8　多态和继承

接下来，我们来看对象最为重要的两个方面：多态（polymorphism）和继承（inheritance）。这两个词很长很深奥，不过正是因为有这两个方面，才使得对象如此有用。我会在下面几节清楚地解释它们的含义。

多态——同一个方法，不同的行为

非常简单，多态是指对于不同的类，可以有同名的两个（或多个）方法。取决于这些方法分别应用到哪个类，它们可以有不同的行为。

例如，假设你要建立一个程序做几何题，需要计算不同形状的面积，比如三角形和正方形。你可以创建两个类，如下：

```
class Triangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def getArea(self):
        area = self.width * self.height / 2.0
        return area

class Square:
    def __init__(self, size):
        self.size = size

    def getArea(self):
        area = self.size * self.size
        return area
```

这是 Triangle 类

都有一个名为 getArea() 的方法

这是 Square 类

Triangle 类和 Square 类都有一个名为 getArea() 的方法。所以，如果分别有这两个类的实例，如下：

```
>>> myTriangle = Triangle(4, 5)
>>> mySquare = Square(7)
```

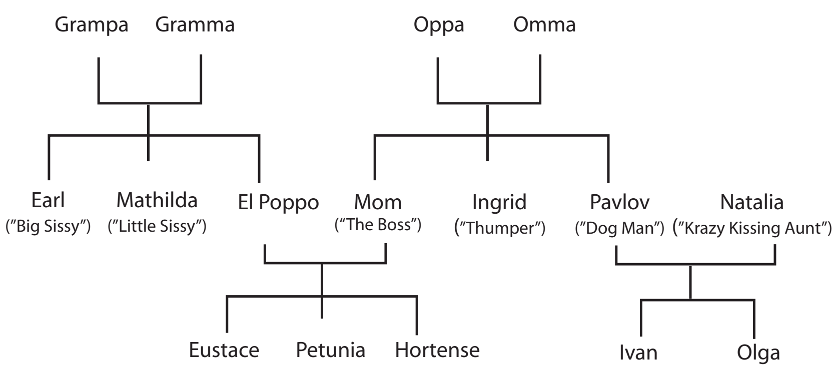
就可以使用 getArea() 分别计算它们的面积：

```
>>> myTriangle.getArea()
10.0
>>> mySquare.getArea()
49
```

这两个形状都使用了方法名 getArea()，不过每个形状中这个方法做的工作不同。这就是一个多态的例子。

继承——向父母学习

在真实的（非编程）世界中，人们可以从他们的父母或者其他亲戚那里继承一些东西。你可以继承一些特征，比如说红头发，或者可以继承像钱和财产之类的东西。



在面向对象编程中，类可以从其他类继承属性和方法。这样就有了类的整个“家族”，这个“家族”中的每个类共享相同的属性和方法。这样一来，每次向“家族”增加新成员时就不必从头开始。

从其他类继承属性或方法的类称为派生类（derived class）或子类（subclass）。可以举一个例子来解释这个概念。

假想我们要建立一个游戏，玩家一路上可以捡起不同的东西，比如食物、钱或衣服。可以建一个类，名为 `GameObject`。`GameObject` 类有 `name` 等属性（例如 coin、apple 或 hat）和 `pickUp()` 等方法（它会把硬币增加到玩家的物品集合中）。所有游戏对象都有这些共同的方法和属性。

然后，可以为硬币建立一个子类。`Coin` 类从 `GameObject` 派生。它要继承 `GameObject` 的属性和方法，所以 `Coin` 类会自动有一个 `name` 属性和 `pickUp()` 方法。`Coin` 类还需要一个 `value` 属性（这个硬币价值多少）和一个 `spend()` 方法（可以用这个硬币去买东西）。

下面来看这些类的代码：

```
class GameObject:
    def __init__(self, name):
        self.name = name

    def pickUp(self, player):
        # put code here to add the object
        # to the player's collection

class Coin(GameObject):
    def __init__(self, value):
        GameObject.__init__(self, "coin")
        self.value = value

    def spend(self, buyer, seller):
        # put code here to remove the coin
        # from the buyer's money and
        # add it to the seller's money
```

定义 GameObject 类

Coin 是的 GameObject 的子类

在 __init__() 中，继承 GameObject 的初始化方法并补充新内容

Coin 类新的 spend() 方法

14.9　未雨绸缪

在上面的例子中，我们并没有在方法中加入任何实际代码，只有一些注释来解释这些方法要做什么。这是一种未雨绸缪的方法，是对以后要增加的内容提前做出计划或提前考虑。具体的代码要取决于游戏如何工作。程序员编写比较复杂的代码时通常就会采用这种做法来组织他们的想法。“空”函数或方法称为代码桩（code stub）。

如果想运行前面的例子，会得到一条错误消息，因为函数定义不能为空。





没错，卡特，不过注释不起作用，因为它们只是给你读的，而不是让计算机来执行。

如果希望建立一个代码桩，可以使用 Python 的 pass 关键字作为一个占位符。代码实际上应该像下面这样：

```
class Game_object:
    def __init__(self, name):
        self.name = name

    def pickUp(self):
        pass
        # put code here to add the object
        # to the player's collection

class Coin(Game_object):
    def __init__(self, value):
        GameObject.__init__(self, "coin")
        self.value = value

    def spend(self, buyer, seller):
        pass
        # put code here to remove the coin
        # from the buyer's money and
        # add it to the seller's money
```

在这两个位置增加 pass 关键字

我不打算再在这一章中给出使用对象、多态和继承的更详细的例子。学习这本书后面的内容时还会看到很多关于对象以及如何使用对象的例子。通过在实际的程序（比如游戏）中使用对象，你会有更深入的理解。

你学到了什么

在这一章，你学到了以下内容。

- 什么是对象。
- 属性和方法。
- 什么是类。
- 创建类的一个实例。
- 特殊方法：`__init__()` 和 `__str__()`。
- 多态。
- 继承。
- 代码桩。

测试题

1. 定义一个新的对象 类型时用什么关键字？
2. 什么是属性？
3. 什么是方法？
4. 类和实例之间有什么区别？
5. 方法中实例引用通常用什么名字？
6. 什么是多态？
7. 什么是继承？

动手试一试

1. 为 `BankAccount` 建立一个类定义。它应该有一些属性，包括账户名（一个字符串）、账号（一个字符串或整数）和余额（一个浮点数），另外还要有一些方法显示余额、存钱和取钱。
2. 建立一个可以挣利息的类，名为 `InterestAccount`。这应当是 `BankAccount` 的一个子类（所以会继承 `BankAccount` 的属性和方法）。`InterestAccount` 还应当有一个对应利息率的属性，另外有一个方法来增加利息。为了力求简单，假设每年会调用一次 `addInterest()` 方法计算利息并更新余额。

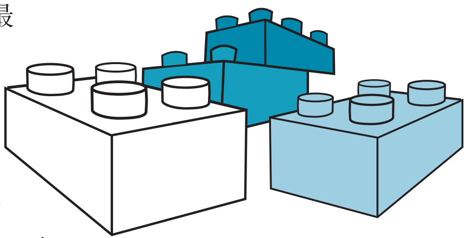
第 15 章

模　　块

这是讨论收集方式的最后一章。前面已经了解了列表、函数和对象，这一章中我们将学习模块。下一章中，我们将使用一个名为 Pygame 的模块开始画一些图形。

15.1　什么是模块

模块就是某个东西的一部分。如果一个东西可以分为几部分，或者你可以很容易地把它分解成多个不同部分，我们就说这个东西是模块化的。乐高（LEGO）积木可能就是模块化最好的例子。可以拿一堆不同的积木，用它们搭建不同的东西。



在 Python 中，模块（module）是包含在一个更大程序中的类似的部分。每个模块或部分都是硬盘上的一个单独的文件。可以把一个大程序分解为多个模块或文件。或者也可以反过来，从一个小的模块开始，逐渐增加其他部分来建立一个大程序。

15.2　为什么使用模块

为什么要那么麻烦地把程序分解为较小的部分呢？要知道我们需要所有这些部分才能让程序正常工作。为什么不直接把所有内容都放在一个大文件中呢？

原因有几个。

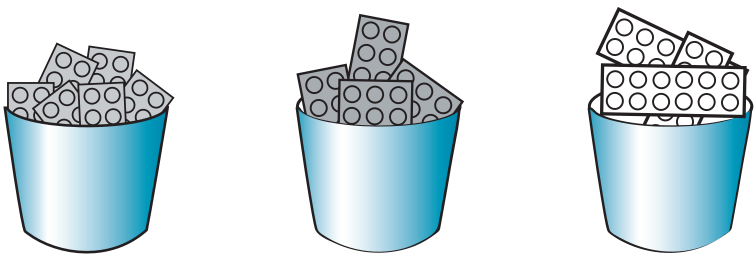
- 这样做文件会更小，因而就能更容易地查找代码。
- 一旦创建模块，这个模块就能在很多程序中使用。这样下一次需要相同的功

能时就不必再从头开始了。

- 并不是所有模块都要使用。模块化意味着你可以使用各部分的不同组合来完成不同的任务，就像利用同样的一组乐高积木可以搭建不同的东西一样。

15.3　积木桶

在关于函数的第 13 章中，我们说过函数就像积木，那么模块可以认为是一桶积木。根据需要，你可以从一个桶中取很多或者很少的积木，也可以有很多桶不同的积木。也许有一桶正方形积木，一桶长方形积木，还有一桶奇形怪状的积木。程序员通常也采用这种方法来使用模块，也就是说，他们会把类似的函数收集在一个模块中。或者他们也有可能把一个项目需要的所有函数收集在一个模块中，就像你会把搭城堡需要的所有积木都放在一个桶中一样。



15.4　如何创建模块

下面来创建模块。模块就是一个 Python 文件，类似代码清单 15-1 中给出的文件。在一个 IDLE 编辑器窗口中键入代码清单 15-1 中的代码，把它保存为 my_module.py。

代码清单 15-1　创建一个模块

```
# this is the file "my_module.py"
# we're going to use it in another program
def c_to_f(celsius):
    fahrenheit = celsius * 9.0 / 5 + 32
    return fahrenheit
```

就这么简单！这样就创建了一个模块！模块中只有一个函数，也就是 `c_to_f()` 函数，它会把温度从摄氏度转换为华氏度。

接下来我们在另一个程序中使用 my_module.py。

15.5　如何使用模块

要使用模块中的某个函数，首先必须告诉 Python 我们想要使用哪些模块。在程序

中包含其他模块的 Python 关键字是 import。可以这样使用：

```
import my_module
```

下面写一个程序来使用我们刚才编写的模块，这里我们想用 c_to_f() 函数完成温度转换。

前面已经了解了如何使用函数并向它传递参数。这里惟一不同的是，函数与主程序不在同一个文件中，而在另外的一个单独的文件中，所以必须使用 import。代码清单 15-2 中的程序使用了我们刚才编写的模块 my_module.py。

代码清单 15-2 使用模块

```
import my_module    <——my_module 包含 c_to_f()
                       函数

celsius = float(raw_input ("Enter a temperature in Celsius: "))
fahrenheit = c_to_f(celsius)
print "That's ", fahrenheit, " degrees Fahrenheit"
```

创建一个新的 IDLE 编辑器窗口，键入这个程序。保存为 modular.py，然后运行这个程序，看看会发生什么。需要把它保存到 my_module.py 所在的同一个文件夹（或目录）下。

能正常工作吗？应该会看到类似下面的结果：

```
>>> ========================= RESTART =========================
>>>
Enter a temperature in Celsius: 34

Traceback (most recent call last):
  File "C:/MyPythonPrograms/modular.py", line 4, in <module>
    fahrenheit = c_to_f(celsius)
NameError: name 'c_to_f' is not defined
```

程序不能正常工作！怎么回事？错误消息指出函数 c_to_f() 没有定义。不过我们很清楚前面已经在 my_module 中定义了这个函数，而且我们确实已经导入了这个模块。

出现这个问题的原因是，在 Python 中指定在其他模块中定义的函数时必须更加具体。解决这个问题的一种方法是把这一行代码

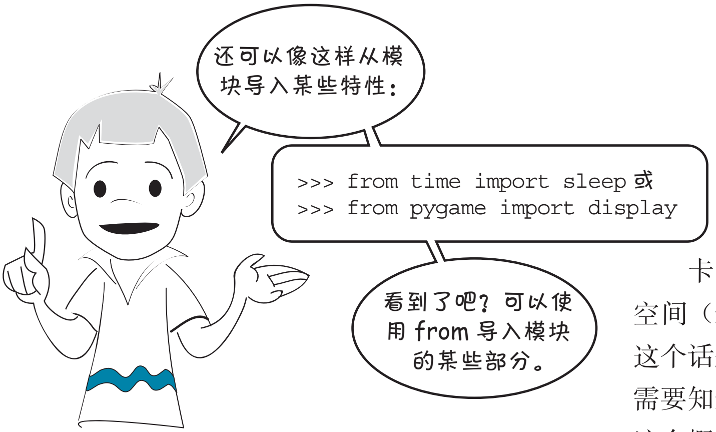
```
fahrenheit = c_to_f(celsius)
```

改为

```
fahrenheit = my_module.c_to_f(celsius)
```

现在我们向 Python 特别指出：c_to_f() 函数在 my_module 模块中。做了这个修改后再试着运行程序，看看能不能正常工作。

15.6　命名空间

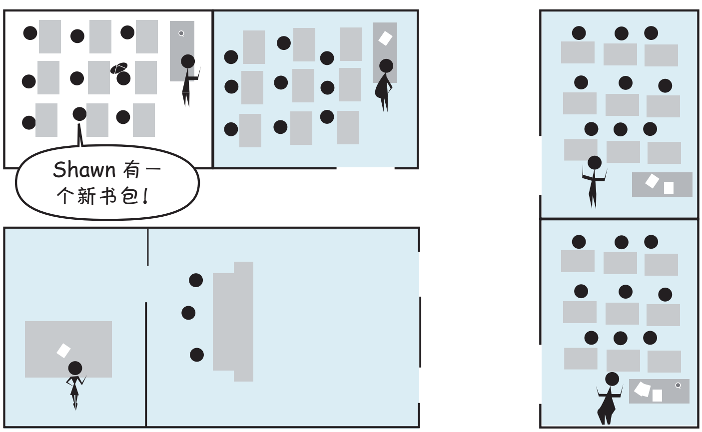


卡特提到的内容与命名空间（namespace）概念有关。这个话题有点复杂，不过确实需要知道，所以现在就来讨论这个概念。

什么是命名空间

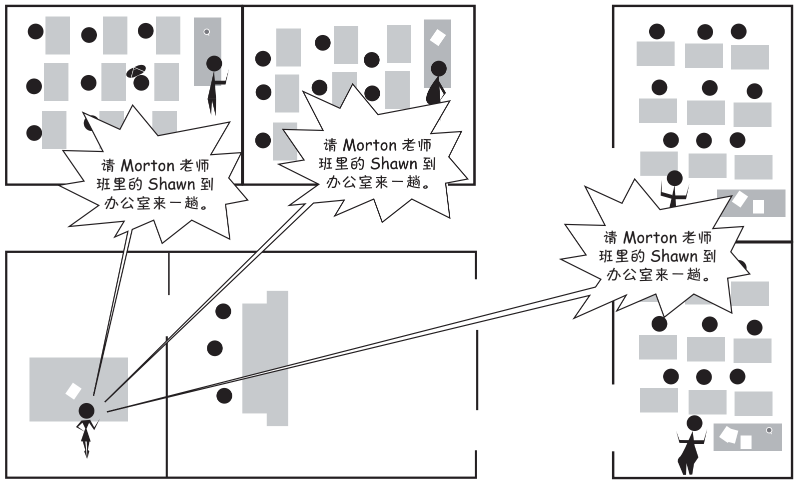
假设在你们学校，你在 Morton 老师的班里，班里有个学生名叫 Shawn。现在 Wheeler 老师教的那个班也有一个名叫 Shawn 的学生。如果你在自己的班里说“Shawn 有一个新书包”时，你们班的所有人都会知道（或者至少他们会认为），你指的是你们班的 Shawn。如果你想说另外那个班的 Shawn 就会说“Wheeler 老师班里的 Shawn”或者“另外那个 Shawn”，或者其他类似的说法。

你们班里只有一个 Shawn，所以你说 Shawn 时，同班的同学就会知道你说的是哪个人。换种说法来讲，在你们班的这个空间里，只有一个名字 Shawn。你们班就是你的命名空间，在这个命名空间里只有一个 Shawn，所以不会有混淆。

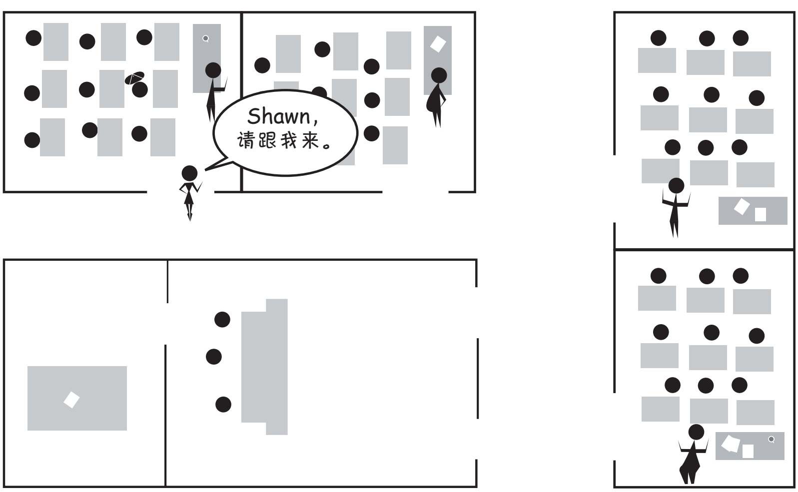


现在，如果校长必须通过学校的广播系统把 Shawn 叫到办公室，她不会说“请

Shawn 到办公室来一趟”。如果她这样做，两个 Shawn 都会出现在他的办公室。对于使用广播系统的校长来说，命名空间是整个学校。这说明，学校的每一个人都会听到这个名字，而不只是一个班的同学。所以她必须更明确地指出她指的是哪一个 Shawn。她必须这样说：“请 Morton 老师班里的 Shawn 到办公室来一趟。”

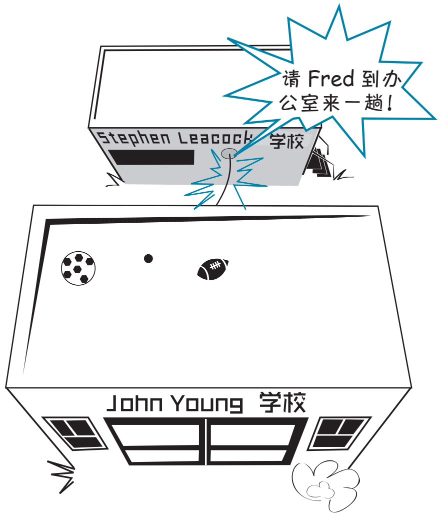


校长还可以用另一种方法找 Shawn，就是走到你们班门口说：“Shawn，请跟我来。”这里只有一个 Shawn 听到，所以校长能找到真正要找的那个 Shawn。在这种情况下，命名空间就只是一个教室，而不是整个学校。



一般来讲，程序员把较小的命名空间（比如你的教室）称作局部命名空间，而较大的命名空间（如整个学校）称为全局命名空间。

导入命名空间



下面假设你们学校（John Young 学校）根本没有一个名叫 Fred 的人。如果校长通过广播系统想找 Fred，她肯定找不到这个人。现在假设与你们学校同在一条街上的另一个学校（Stephen Leacock 学校）正在进行部分校舍维修，这个学校把一个班级临时搬到你们学校的活动房里上课。在这个班里，恰好有一个学生名叫 Fred。不过这个活动房还没有连上广播系统。如果校长找 Fred，肯定还是找不到。但是，如果她把这个新的活动房连入广播系统，然后再找 Fred，就会找到 Stephen Leacock 学校的 Fred。

连接另一个学校的活动房屋，这在 Python 中就像导入一个模块。导入了模块，就可以访问这个模块中的所有名字，包括所有变量、函数以及对象。

导入模块的含义与导入一个命名空间是一样的。导入模块时，就导入了命名空间。

导入命名空间（模块）有两种方法。可以这样做：

```
import StephenLeacock
```

如果这样做，StephenLeacock 仍然是一个单独的命名空间。你可以访问这个命名空间，但是在使用之前必须明确地指定想要哪一个命名空间。所以校长必须这样做：

```
call_to_office(StephenLeacock.Fred)
```

如果校长想找到 Fred，除了名字（Fred）外，她还必须给出命名空间（Stephen Leacock）。在前面的温度转换程序中就是这样做的。

为了让这个程序正常工作，我们写了这样一行代码：

```
fahrenheit = my_module.c_to_f(celsius)
```

这里指定了命名空间（my_module）以及函数名（c_to_f）。

用 from 导入

导入命名空间的另一种方法是：

```
from StephenLeacock import Fred
```

如果校长这样做，会把 StephenLeacock 的名字 Fred 包含到她的命名空间中，现在就可以这样找到 Fred：

```
call_to_office(Fred)
```

因为 Fred 现在就在校长的命名空间中，所以她不必再去 StephenLeacock 命名空间找 Fred。

在这个例子中，校长只是从 StephenLeacock 把名字 Fred 导入她的局部命名空间中。如果她想导入所有人，可以这样做：

```
from StephenLeacock import *
```

在这里，星号（*）表示全部。不过她必须当心，如果 Stephen Leacock 学校与 John Young 学校有同名的学生，就会出现混乱了。

太难了

到目前为止，你可能对命名空间的概念还是不太清楚。不用担心！通过完成后面几章的例子，你会越来越明白。后面需要导入模块时，我都会清楚地解释要做什么。

15.7 标准模块

我们已经知道了如何创建和使用模块，是不是总是必须编写我们自己的模块？并不是这样！这正是 Python 的妙处之一。

Python 提供了大量标准模块，可以用来完成很多工作，比如查找文件、报时（或计时）、生成随机数，以及很多其他功能。有时，人们说 Python“配有电池”，指的就是 Python 的所有标准模块。这称为 Python 标准库。

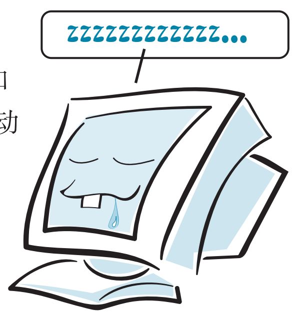
为什么这些内容必须放在单独的模块中呢？嗯，不是非得这样，不过设计 Python 的人认为这样会更高效。否则，每个 Python 程序都必须包含所有可能用到的函数。通过建立单独的模块，就只需包含你真正需要的那些函数。

当然，有些内容（如 print、for 和 if-else）是 Python 的基本命令，所以这些基本命令不需要一个单独的模块，它们都在 Python 的主要部分中。

如果 Python 没有提供合适的模块来完成你想做的工作（如建立一个图形游戏），可以下载另外一些插件模块，它们通常都是免费的！我们在这本书里就包含了一些这样的插件模块，如果使用这本书网站上的安装程序，就会同时安装这些模块。或者，你也完全可以单独安装。

下面来看几个标准模块。

time



利用 time 模块，能够获取你的计算机时钟的信息，如日期和时间。还可以利用它为程序增加延迟。（有时计算机动作太快，你必须让它慢下来。）

time 模块中的 sleep() 函数可以用来增加一个延迟，也就是说，可以让程序等待一段时间，什么也不做。这就像让你的程序睡眠，正是这个原因，这个函数名叫

sleep()。可以告诉它你要它睡多长时间（多少秒）。

代码清单 15-3 中的程序展示了 sleep() 函数如何工作。键入这个程序，保存并运行，看看会发生什么。

代码清单 15-3 让程序睡眠

```
import time
print "How",
time.sleep(2)
print "are",
time.sleep(2)
print "you",
time.sleep(2)
print "today?"
```

要注意，调用 sleep() 函数时，必须在前面加上 time.。这是因为，尽管我们已经用 import 导入了 time，但是并没有让它成为主程序命名空间的一部分。所以每次想要使用 sleep() 函数时，都必须调用 time.sleep()。

如果试图这样做：

```
import time
sleep(5)
```

这是不行的，因为 sleep() 并不在我们的命名空间中。我们会得到这样一条错误消息：

```
NameError: name 'sleep' is not defined
```

不过如果这样导入：

```
from time import sleep
```

就会告诉 Python，“在 time 模块中寻找名为 sleep 的变量（或者函数或对象），把它包含到我的命名空间中。”现在就可以直接使用 sleep() 函数，而不需要再在前面加上 time. 了：

```
from time import sleep
print 'Hello, talk to you again in 5 seconds...'
sleep(5)
print 'Hi again'
```

如果想要得到这种将名字导入局部命名空间带来的方便（这样就无需每次都指定模块名），但是又不知道需要模块中的哪些名字，就可以使用星号（*）把所有名字都导入到我们的命名空间里：

```
from time import *
```

* 表示“全部”，这样就会从模块导入所有可用的名字。使用这个命令必须特别当心。如果在我们的程序中创建了一个名字，而它与 time 模块中的一个名字相同，就会出现冲突。用 * 导入所有名字不是最佳做法，最好只导入你真正需要的部分。

还记得第 8 章代码清单 8-6 中的倒计时程序吗？现在你应该知道那个程序中 time.sleep(1) 的作用了吧。

随机数

random 模块用于生成随机数。这在游戏和仿真中非常有用。

下面试着在交互模式中使用 random 模块：

```
>>> import random
>>> print random.randint(0, 100)
4
>>> print random.randint(0, 100)
72
```

每次使用 random.randint() 时，会得到一个新的随机整数。由于我们为它传递的参数是 0 和 100，所以得到的整数会介于 0 到 100 之间。我们在第 1 章的猜数程序中就是使用 random.randint() 来创建秘密数。

如果你想得到一个随机的小数，可以使用 random.random()。不用在括号里放任何参数，因为 random.random() 总是会提供一个介于 0 到 1 之间的数：

```
>>> print random.random()
0.270985467261
>>> print random.random()
0.569236541309
```

如果你想得到其他范围内的一个随机数，比如说 0 到 10 之间，只需要将结果乘以 10：

```
>>> print random.random() * 10
3.61204895736
>>> print random.random() * 10
8.10985427783
```

你学到了什么

在这一章，你学到了以下内容。

- 什么是模块。
- 如何创建模块。
- 如何在另一个程序中使用模块。
- 什么是命名空间。
- 局部和全局命名空间和变量是什么意思。

- ❒ 如何把其他模块中的名字包含到你的命名空间中。

另外你还了解了几个 Python 标准模块的例子。

测试题

1. 使用模块有哪些好处？
2. 如何创建模块？
3. 使用模块时所用的 Python 关键字是什么？
4. 导入模块等同于导入一个 ___________。
5. 要导入 time 模块从而能访问这个模块中的所有名字（也就是所有变量、函数和对象），有哪两种方法？

动手试一试

1. 编写一个模块，包含第 13 章“动手试一试”中的“用大写字母打印名字”函数。然后编写一个程序导入这个模块，并调用这个函数。
2. 修改代码清单 15-2 中的代码，把 c_to_f() 包含到主程序的命名空间里。也就是说，修改这个代码，从而可以写：

```
fahrenheit = c_to_f(celsius)
```

而不是

```
fahrenheit = my_module.c_to_f(celsius)
```

3. 编写一个小程序，生成 1 到 20 之间的 5 个随机整数的列表，并打印出来。
4. 编写一个小程序，要求它工作 30 秒，每 3 秒打印一个随机小数。

第 16 章 图形

你已经了解了计算机编程的很多基本要素：输入和输出、变量、判断、循环、列表、函数、对象和模块。希望你能为掌握这些知识感到高兴！现在该利用 Python 和编程做点更有意思的事情了。

这一章中，你会学习如何在屏幕上画图，比如直线、形状、颜色，还会谈到一点动画。这会帮助我们在后面的几章中真正开发游戏和其他程序。

16.1 寻求帮助——Pygame

要让图形（和声音）在你的计算机上起作用，这可能有点复杂。这涉及操作系统和你的图形卡，还需要大量底层代码（目前我们还不想考虑这些代码）。所以我们将使用一个名为 Pygame 的 Python 模块来提供帮助，让问题更简单一些。

要让游戏在不同计算机和操作系统上都能工作，所需要的图形和其他内容都可以利用 Pygame 来创建，而不必了解每个系统的烦琐细节。Pygame 是免费的，这本书提供了 Pygame 的一个版本。如果你使用这本书的安装程序来安装 Python，应该已经同时安装了 Pygame。否则，必须单独安装 Pygame，可以从 Pygame 网站（www.pygame.org）得到。

16.2 Pygame 窗口

开始绘制图形时首先需要建立一个窗口。代码清单 16-1 显示了一个非常简单的程序，它只是创建了一个 Pygame 窗口。

代码清单 16-1 创建一个 Pygame 窗口

```
import pygame
pygame.init()
screen = pygame.display.set_mode([640, 480])
```

试着运行这个程序。你看到了什么？取决于你使用的操作系统，你可能会看到屏幕非常迅速地弹出了一个窗口（填充为黑色）。你也可能发现弹出的窗口无法关闭。这是怎么回事？

嗯，Pygame 的作用就是为了建立游戏。游戏本身不做任何事情，只是与玩家交互。所以 Pygame 有一个事件循环（event loop）不断检查用户在做什么，比如按键、移动鼠标或关闭窗口。Pygame 程序需要有个事件循环一直运行。在我们的第一个 Pygame 程序中，并没有启动事件循环，所以程序没有正常运行。

要想保持 Pygame 事件循环一直运行，一种方法是使用 `while` 循环。我们希望这个循环可以随着程序的运行一直运行下去。因为 Pygame 程序通常没有菜单，所以用户要关闭程序的话，需要使用窗口右上角的 ×（Windows），或者左上角的关闭按钮（MacOS）。对 Linux 系统来说，关闭按钮的位置取决于使用的窗口管理器和 GUI 框架。但如果你在使用 Linux，我想你应该知道怎么关闭窗口。

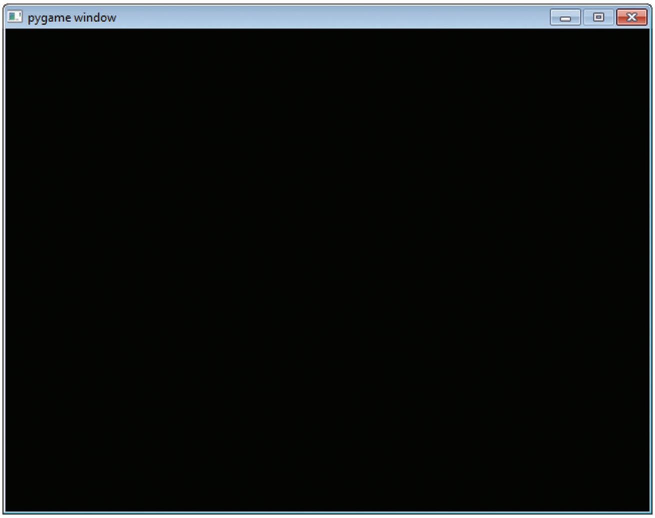
下面的代码清单打开了一个 Pygame 窗口，并在用户关闭它之前一直保持运行状态：

代码清单 16-2 使 Pygame 窗口正确工作

```
import pygame
pygame.init()
screen = pygame.display.set_mode([640, 480])
running  = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
             running = False
pygame.quit()
```

运行以上代码，你会看到一个正常工作的 Pygame 窗口，该窗口会在你尝试关闭时关闭。

`while` 循环中的语句到底是如何工作的呢？它使用了 Pygame 的事件循环。但这个话题我们留到第 18 章再讲，届时我们会讲解 Pygame 中的事件。



16.3 在窗口中画图

现在我们有了一个 Pygame 窗口，在我们优雅地将它关闭之前它会一直打开。代码清单 16-2 的第 3 行中的 [640, 480] 是窗口的大小，表示 640 像素宽、480 像素高。下面就在这里面画一些图形。按照代码清单 16-4 修改你的程序。

代码清单 16-3 画一个圆

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255,255,255])
pygame.draw.circle(screen, [255,0,0],[100,100], 30, 0)
pygame.display.flip()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

用白色背景填充窗口

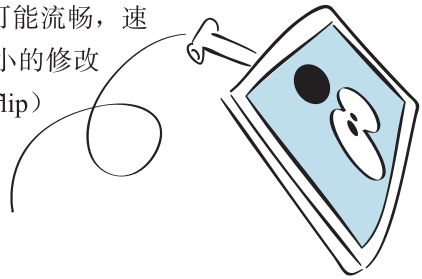
画一个圆

增加这 3 行代码

把你的监视器翻过来……开玩笑的，别当真！

什么是翻转

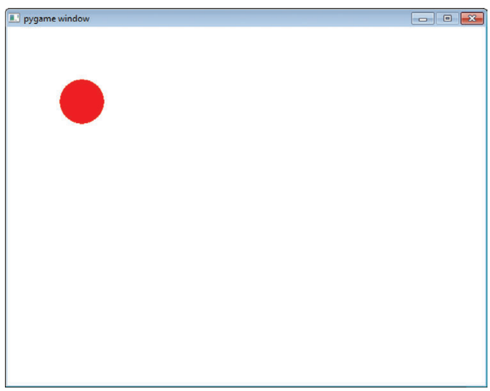
对于 Pygame 窗口中显示的所有内容，Pygame 中的显示对象（我们的显示对象名为 screen，这在代码清单 16-3 的第 3 行创建）都会有这些内容的两个副本。这样做的原因是，开始动画时，我们希望让动画尽可能流畅，速度尽可能快。所以不必在每次对图形做一个小小的修改时都更新显示，可以做很多修改后再“翻转”（flip）到图形的新版本。这样就会一次显示所有这些修改，而不是一个接一个地出现。这样一来，我们就不会显示出只画了一半的圆（或外星人，或者其他任何东西）。



可以把这两个副本当作一个“当前屏”和一个“下一屏”。当前屏就是我们现在看到的，下一屏是完成“翻转”之后看到的。做完“下一屏”上的所有修改后，再翻转到下一屏，就能看到这些改变。

如何建立一个圆

运行代码清单 16-3 中的程序时，应该能看到如右图这样，靠近窗口左上角有一个红色的圆。



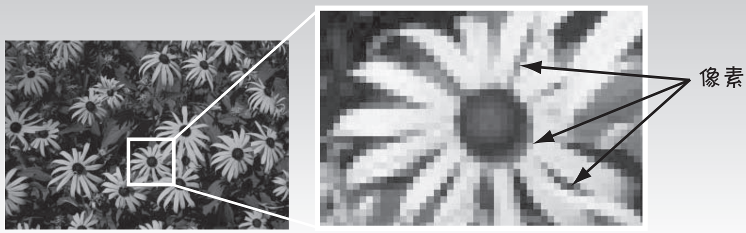
毫不奇怪，pygame.draw.circle() 函数会画一个圆。必须告诉它以下 5 件事。

- 在哪个表面（surface）画这个圆。（在这里，要在第 3 行定义的表面上画圆，名为 screen，这就是显示表面。）
- 用什么颜色来画。（在这里要用红色，对应的值为 [255, 0, 0]。）
- 在什么位置画。（在这里要位于 [100, 100]，这表示从左上角向下 100 像素再向右 100 像素的位置）。
- 圆的大小。（这里是 30，这是圆的半径，也就是圆心到外围边界的距离，单位是像素。）
- 线宽。（如果 width = 0，圆是完全填充的，这里就采用了完全填充。）

下面再来更详细地学习这 5 个方面。

术语箱

像素（pixel）这个词是“图像元素”（picture element）的简写。这表示屏幕上或图像中的一点。如果在一个图像浏览器中查看图片，充分放大（让图像非常大），就可以看到单个的像素。下面是一张照片的正常视图和放大视图，在放大视图中可以看到像素。





哇，你眼力真好！这些小线条实际上就是像素行。一般的计算机屏幕可能有 768 行像素，每行有 1024 个像素。我们就会说这个屏幕“分辨率是 1024×768”。有些屏幕的像素更多，有些可能比这要少。

Pygame 表面

在实际生活中如果我让你画一幅画，你可能会先问“我在哪儿画？”在 Pygame 中，我们要在一个表面上画图。显示表面就是我们在屏幕上看到的表面，也就是代码清单 16-3 中的 `screen`。不过 Pygame 程序可以有多个表面，可以把图像从一个表面复制到另一个表面。还可以对表面做一些处理，比如旋转表面或者调整它们的大小（让它们更大或更小）。

前面提到过，显示表面有两个副本。按软件术语来讲，我们说显示表面是双缓冲的（double-buffered）。正是因为这个原因，我们不会在屏幕上看到只画了一半的形状和图像。我们会在缓冲区里画圆、外星人或者任何东西，然后“翻转”显示表面，来显示已经完全绘制的图像。

Pygame 中的颜色

Pygame 中使用的颜色系统是很多计算机语言和程序中通用的系统，称为 RGB。这里的 R、G 和 B 分别代表红、绿和蓝。

你可能在自然课上已经学过，通过结合或混合光的三原色（红、绿和蓝）可以得到任何颜色。计算机上也采用了同样的做法。每个颜色（红、绿和蓝）对应一个从 0 到 255 的数。由一个包含 3 个整数的列表来给出颜色，每个数的范围在 0 到 255 之间。如果所有数都是 0，就没有任何颜色，这就是全黑，所以会得到黑色。如果三个数都是 255，会将 3 种颜色以最大亮度混合在一起，这就是白色。如果颜色是 [255, 0, 0]，这就是纯红色，没有绿色和蓝色。纯绿就是 [0, 255, 0]，纯蓝是 [0, 0, 255]。如果所有 3 个数都一样，比如 [150, 150,150]，你会得到某种灰度。数字越小，灰度就越深，数字越大，灰度就越浅。

颜色名

Pygame 提供了一个命名颜色列表，如果你不想使用 [R, G, B] 记法，就可以使用这些命名颜色。定义好的颜色名有 600 多个。我不想在这里全部列出，不过如果你想看看到底有哪些颜色，可以在你的硬盘上搜索一个名为 colordict.py 的文件，然后在文本编辑器中打开这个文件。

如果你想使用这些颜色名，必须在程序最前面增加下面这行代码：

```
from pygame.color import THECOLORS
```

然后，使用某个命名颜色时，可以这样做（我们的画圆例子中就是这样做的）：

```
pygame.draw.circle(screen, THECOLORS["red"],[100,100], 30, 0)
```

到底怎么回事？



为什么是 255 呢？每个三原色（红、绿、蓝）都有 0 到 255 共 256 个不同的值。256 这个数字有什么特别的呢？为什么不是 200、300 或者 500 呢？

8 位总共正好能表示 256 个不同的值，也就是由 1 和 0 构成的八位数的所有可能的组合。8 位也称为一个字节，字节是有自己的地址的最小内存块。计算机就是利用地址来查找某段内存的。

这就像在街道上一样。你的房子或公寓有一个地址，不过你的房间没有地址。房子就是街道上最小的“可寻址单位”。字节则是计算机内存中最小的“可寻址单位”。

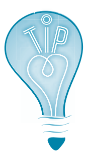
也可以用 8 位以上来表示每种颜色，不过，8 位之后可用的位数可能就到 16 位（2 个字节）了，因为不完整的字节使用起来会不太方便。事实证明，根据人眼识别颜色的方式，用 8 位来表示实际可见的颜色完全足够了。

由于有 3 个值（红、绿、蓝），每个值有 8 位，总共就是 24 位，所以这种表示颜色的方法也称为“24 位颜色表示法”。对每个像素使用 24 位，每个三原色分别使用 8 位。

如果你想试验一下，看看红色、绿色和蓝色如何结合来生成不同的颜色，可以试试 colormixer.py 程序，运行本书的安装程序时会把这个程序放在 \examples 文件夹下。利用这个程序，你可以尝试红、绿、蓝的任意组合，看看能得到什么颜色。

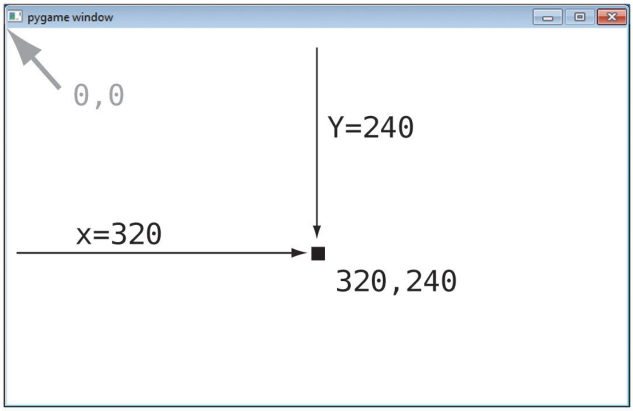
位置——屏幕坐标

如果想在屏幕上画个东西或者放上一个东西，需要指定这个东西应当放在屏幕上的哪个位置。这里有两个数：一个对应 x 轴（水平方向），还有一个对应 y 轴（垂直方向）。在 Pygame 中，这两个数从窗口左上角的 [0, 0] 坐标开始。



看到类似 [320, 240] 的一对数时，要知道第一个数表示水平方向，也就是相对于左边界的距离。第二个数表示垂直方向，也就是相对于顶边的距离。在数学和编程中，字母 x 通常用来表示水平距离，y 常用来表示垂直距离。

我们建立了一个 640 像素宽、480 像素高的窗口。如果希望在窗口正中间画圆，需要在 [320, 240] 上绘制。这个位置离左边界 320 像素，离上边界 240 像素。



下面尝试在窗口中间画圆。运行代码清单 16-4 中的程序。

代码清单 16-4　把圆放在窗口中间

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
pygame.draw.circle(screen, [255,0,0],[320,240], 30, 0)
pygame.display.flip()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

将 [100, 100] 改为 [320, 240]

这里使用坐标 [320, 240] 作为圆心。把运行代码清单 16-3 的结果与运行代码清单 16-4 时看到的结果做个比较，看看有什么差别。

形状大小

使用 Pygame 的 draw 函数画形状时，必须指定形状的尺寸。对于圆来说，只有一个尺寸，也就是半径。而像矩形之类的形状，则必须指定长和宽。

Pygame 有一种特殊的对象，名为 rect（这是“rectangle”（矩形）的简写），用来定义矩形区域。rect 要使用左上角坐标、宽和高来定义：

```
Rect(left, top, width, height)
```

这里同时定义了位置和大小。下面是一个例子：

```
my_rect = Rect(250, 150, 300, 200)
```

这会创建一个矩形，它的左上角距离窗口左边界 250 像素，距离窗口上边界 150 像素，宽为 300 像素，高为 200 像素。下面来试试看。

用下面这行代码替换代码清单 16-4 中的第 5 行，看看结果是什么：

```
pygame.draw.rect(screen, [255,0,0], [250, 150, 300, 200], 0)
```

矩形的颜色　　矩形的位置和大小　　线宽（或填充）

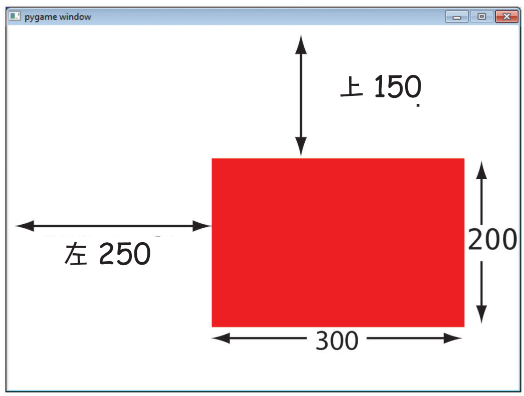
矩形的位置和大小可以是一个简单的数字列表（或元组），也可以是一个 Pygame 的 `Rect` 对象。所以还可以把前面一行替换为下面这两行代码：

```
my_list = [250, 150, 300, 200]
pygame.draw.rect(screen, [255,0,0], my_list, 0)
```

或者

```
my_rect = pygame.Rect(250, 150, 300, 200)
pygame.draw.rect(screen, [255,0,0], my_rect, 0)
```

这就是最后得到的矩形。我增加了一些尺寸标注来说明每个数字分别表示什么含义。



注意这里只向 `pygame.draw.rect` 传递了 4 个参数，因为 `rect` 用一个参数就表示了位置和大小。在 `pygame.draw.circle` 中，位置和大小分别由两个不同的参数表示，所以需要传递 5 个参数。

像（Pygame）程序员一样思考

如果用 Rect(left, top, width, height) 创建一个矩形，还可以使用其他一些属性来移动和对齐这个 Rect：

- 4 条边：top、left、bottom、right
- 4 个角：topleft、bottomleft、topright、bottomright
- 每条边的中点：midtop、midleft、midbottom、midright
- 中心：center、centerx、centery
- 尺寸：size、width、height

这些属性只是为了提供方便。所以，如果你想移动一个矩形，让它的中心位于某个点，不必得出上坐标和左坐标分别是什么；可以直接访问中心位置。



线宽

画形状时最后需要指定的一点是线的粗细。在之前的例子中，我们使用的线宽都是 0，这会填充整个形状。如果使用不同的线宽，会看到形状的轮廓。

试着把线宽改为 2：

```
pygame.draw.rect(screen, [255,0,0], [250, 150, 300, 200], 2)
```

把它设置为 2

试试看有什么结果。再试试其他线宽。

现代艺术

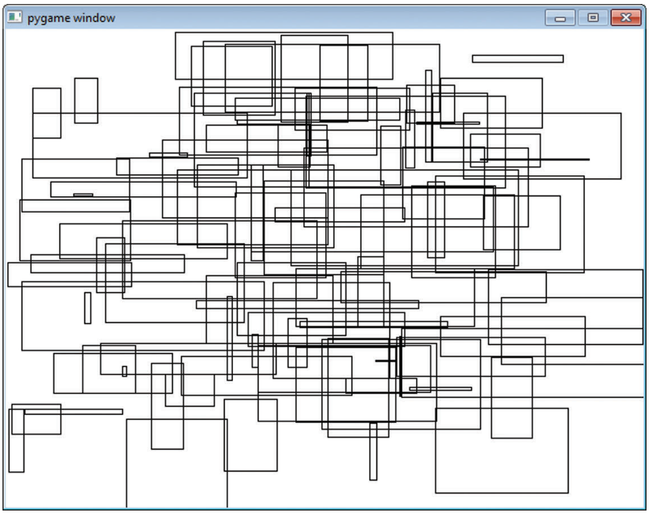
想不想让计算机生成某种现代艺术？玩玩呗，试试代码清单 16-5。也可以在代码清单 16-4 的基础上做些修改，或者干脆从头开始键入。

代码清单 16-5　使用 draw.rect 实现艺术创作

```
import pygame, sys, random
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
for i in range (100):
    width = random.randint(0, 250)
    height = random.randint(0, 100)
```

```
    top = random.randint(0, 400)
    left = random.randint(0, 500)
    pygame.draw.rect(screen, [0,0,0], [left, top, width, height], 1)
pygame.display.flip()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

运行这个程序，看看会得到什么。应该能得到如下图所示的结果：



你明白这个程序是怎么工作的吗？它会随机画 100 个大小不等、位置不同的矩形。为了让它更有“艺术性”，再增加一些颜色，另外将线宽也设为随机，如代码清单 16-7 所示。

代码清单 16-6 带颜色的现代艺术

```
import pygame, sys, random
from pygame.color import THECOLORS
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
for i in range (100):
    width = random.randint(0, 250); height = random.randint(0, 100)
    top = random.randint(0, 400); left = random.randint(0, 500)
    color_name = random.choice(THECOLORS.keys())   <- 现在不用操心这行代码是如何工作的了
    color = THECOLORS[color_name]
    line_width = random.randint(1, 3)
    pygame.draw.rect(screen, color, [left, top, width, height], line_width)
pygame.display.flip()
```

```
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

运行这个程序时，每次你都会看到不同的东西。如果有看着不错的，可以给它起个富有想象力的名字，比如“机器之声”，看看能不能把它卖到你们当地的美术馆！

16.4 单个像素

有时我们并不想画一个圆或矩形，而是希望画单个的点或像素。比如，我们要创建数学程序，想画一条正弦曲线。

别担心，放松点！

如果你不知道什么是正弦曲线也不用担心。学习本章的内容，只要知道这是一种波浪形的曲线就可以了。

另外也不用担心后面几个示例程序中的数学公式，按代码清单原样键入这些程序就行。这些公式只是为了保证得到大小合适的波浪形状，能够放入我们的 Pygame 窗口中。



由于没有 pygame.draw.sinewave() 这种方法，所以我们必须利用单个的点来画这样一条曲线。一种方法是画很小的圆或矩形（圆或矩形的大小只有一个或两个像素）。代码清单 16-7 显示了用矩形画出来的曲线是什么样的。

代码清单 16-7　用大量很小的矩形画曲线

```
import pygame, sys
import math
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
for x in range(0, 640):
```

导入数学函数，包括 sin()（指向 import math）

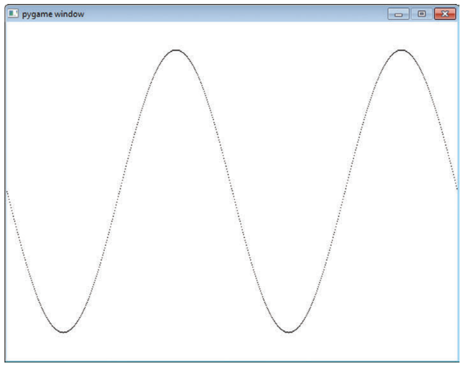
从左到右循环（x = 0 到 639）（指向 for x in range(0, 640):）

```
    y = int(math.sin(x/640.0 * 4 * math.pi) * 200 + 240)
    pygame.draw.rect(screen, [0,0,0],[x, y, 1, 1], 1)
pygame.display.flip()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

计算每个点的y坐标（垂直坐标）

使用小矩形来画点

运行这个程序时会看到如右图所示的结果。



每个点都是宽和高分别为 1 像素的矩形。注意我们使用的线宽为 1，而不是 0。如果使用线宽 0，什么都不会显示，因为这样一个矩形没有“中间部分”可以填充。

连接多个点

如果你看得确实很仔细，可能会注意到，这个正弦曲线并不是连续的，中间的点之间存在空格。这是因为，在正弦曲线比较陡的部分，我们必须上移（或下移）3 个像素而向右只移动 1 个像素。而且由于我们画的是单个的点，而不是线，所以没有什么来填充它们之间的间隔。

下面还是做同样的工作，不过现在要用一条短线把各个点连接起来。Pygame 有一个画线的方法，另外还有一种方法可以在一系列点之间画线（类似于“连接多个点”）。这个方法是 pygame.draw.lines()，它需要下面这 5 个参数。

- 画线的表面（surface）。
- 颜色（color）。
- 是否要画一条线将最后一个点与第一个点相连接，使形状闭合（closed）。我们不希望正弦曲线闭合，所以对我们来说，这个参数是 False。
- 要连接的点的列表（list）。
- 线宽（width）。

所以，在我们的正弦曲线例子中，pygame.draw.lines() 方法是这样的：

```
pygame.draw.lines(screen, [0,0,0],False, plotPoints, 1)
```

在 for 循环中，并没有画出各个点，我们只是创建了 draw.lines() 将要连接的点列表。然后在 for 循环之外调用一次 draw.lines()。整个程序如代码清单 16-8 所示。

代码清单 16-8　一条完美连接的正弦曲线

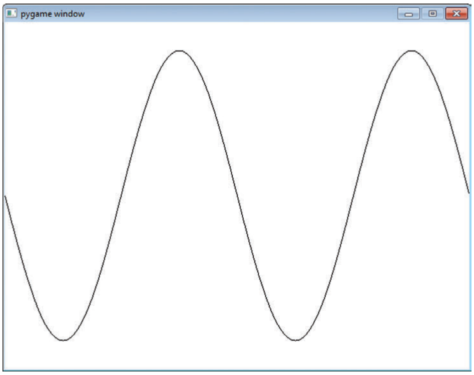
```
import pygame, sys
import math
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
plotPoints = []
for x in range(0, 640):
    y = int(math.sin(x/640.0 * 4 * math.pi) * 200 + 240)
    plotPoints.append([x, y])
pygame.draw.lines(screen, [0,0,0],False, plotPoints, 1)
pygame.display.flip()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

计算每个点的 y 坐标

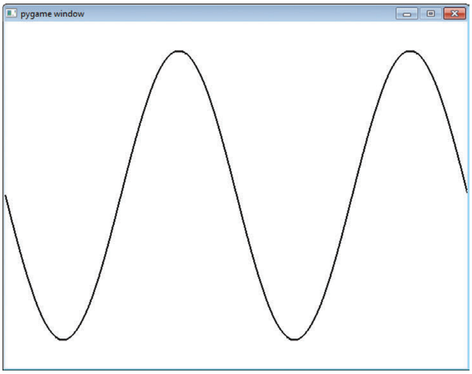
将各个点添加到列表

用 draw.lines() 函数画出整条曲线

现在运行这个程序时，可以看到如右图所示的曲线。



这就好多了，点与点之间不再有间隔。如果把线宽增加到 2，看上去会更好，如右图所示。



再来连接多个点

还记得小时候玩过的连数字画图吗？这里给出一个 Pygame 版本。

代码清单 16-9 中的程序使用了 `draw.lines()` 函数和一个点列表来创建图形。要想看到这个神秘的图片，必须键入代码清单 16-9 中的程序。这一次没有捷径可走！我们没有把这个程序包含在 \examples 文件夹中，如果你想看到这个神秘的图片，就必须自己键入。不过键入所有这些数字可能有点乏味，所以你可以在 \examples 文件夹或网站上的一个文本文件中找到这个 dots 列表。

代码清单 16-9　连连看神秘图片

```
import pygame, sys
pygame.init()

dots = [[221, 432], [225, 331], [133, 342], [141, 310],
        [51, 230],  [74, 217],  [58, 153],  [114, 164],
        [123, 135], [176, 190], [159, 77],  [193, 93],
        [230, 28],  [267, 93],  [301, 77],  [284, 190],
        [327, 135], [336, 164], [402, 153], [386, 217],
        [409, 230], [319, 310], [327, 342], [233, 331],
        [237, 432]]

screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
pygame.draw.lines(screen, [255,0,0],True, dots, 2)      ← 这一次 closed=True
pygame.display.flip()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

逐点绘制

下面再来考虑逐点绘制。如果我们只想改变一个像素的颜色，画一个小圆或矩形就会有点傻。你可以不使用 `draw` 函数，而是利用 `Surface.set_at()` 方法访问一个表面上的单个像素。你要指出希望设置哪个像素，以及要设置成什么颜色：

```
screen.set_at([x, y], [0, 0, 0])
```

如果在我们的正弦曲线例子中使用这行代码（放在代码清单 16-7 的第 8 行），看上去与使用 1 个像素宽的矩形画出的结果完全相同。

还可以用 `Surface.get_at()` 方法检查一个像素设置为什么颜色。只需要传入

你想要检查的那个像素的坐标，比如：pixel_color = screen.get_at([320, 240])。在这个例子中，screen 是表面的名字。

16.5 图像

在屏幕上画形状、线和单个像素只是制作图形的一种方式。有时我们还想用从别处得来的图片、可能是数码照片、从网上下载的图片或者在图像编辑程序中创建的图片。在 Pygame 中，使用图像最简单的方法就是利用 image 函数。

下面来看一个例子。我们要显示一个图像，如果你用本书的安装程序安装了 Python，这个图像已经在你的硬盘上了。安装程序会在 \examples 文件夹中创建一个 images 子文件夹，这个程序中要使用的文件是 beach_ball.png。所以，如果你的系统是 Windows，你会在这里找到这个文件：

c:\Program Files\helloworld\examples\images\beach_ball.png。

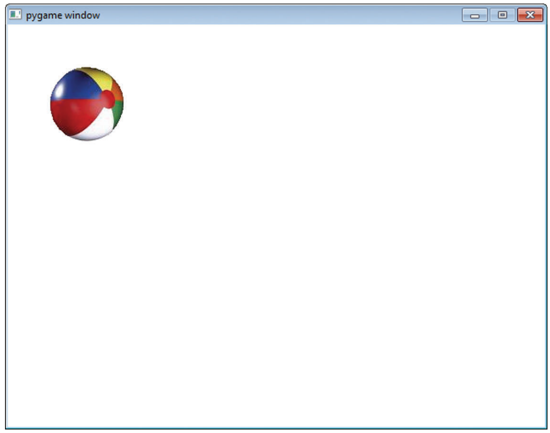
完成这些例子时，需要把 beach_ball.png 文件复制到保存 Python 程序的同一位置上。这样一来，程序运行时 Python 就能很容易地找到这个文件。把 beach_ball.png 文件放在正确的位置上后，键入代码清单 16-10 中的程序，试着运行这个程序。

> 如果没有使用这本书的安装程序，可以从本书网站（www.helloworldbook2.com）下载 beach_ball.png。

代码清单 16-10 在 Pygame 窗口中显示沙滩球图像

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
my_ball = pygame.image.load("beach_ball.png")
screen.blit(my_ball, [50, 50])
pygame.display.flip()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

只有这几行代码是新加的



运行这个程序时，会看到一个沙滩球的图像显示在 Pygame 窗口的左上角附近，如右图所示。

代码清单 16-10 中，只有第 5 行和第 6 行代码是新加的代码。所有其他代码都在代码清单 16-3 到代码清单 16-9 中见过。我们把先前例子中的 draw 代码替换为从硬盘加载图像并显示图像的代码。

第 5 行中，pygame.image.load() 函数从硬盘加载一个图像，并创建一个名为 my_ball 的对象。my_ball 对象是一个表面（前面讨论过表面）。不过我们看不到这个表面，它只在内存中。我们唯一能看到的表面是显示表面，名为 screen（这在第 3 行创建）。第 6 行把 my_ball 表面复制到 screen 表面上。然后像前面一样，通过 display.flip() 调用使它可见。



没关系的，卡特。很快我们就可以移动这个球了！

你可能已经注意到代码清单 16-10 第 6 行有一个看上去很有趣的东西：screen.blit()。blit 是什么意思？请从“术语箱”找出答案。

在 Pygame 中，我们将像素从一个表面复制或块移到另一个表面，这里就是将像素从 my_ball 表面复制到 screen 表面。

术语箱

完成图形编程时，将像素从一个地方复制到另一个地方是很常见的（比如从变量复制到屏幕，或者从一个表面复制到另一个表面）。像素复制在编程中有一个特殊的名字，叫做块移（blitting）。我们说将一个图像（或图像的一部分，或者只是一些像素）从一个地方“块移”到另一个地方。这只是“复制”的一种有趣的说法，不过看到“块移”时，你就会知道复制的是像素而不是其他内容。

在代码清单 16-10 的第 6 行，我们把沙滩球图像块移到位置 50, 50，这表示距窗口左边界 50 像素，距上边界 50 像素。处理 surface 或 rect 时，这会设置图像左上角的位置。所以沙滩球的左边距离窗口左边界有 50 像素，沙滩球的顶边距离窗口上边界也是 50 像素。

16.6 动起来

既然可以把图形放在 Pygame 窗口中，就让它动起来吧。没错，我们要做一些动画了！计算机动画实际上就是把图像（像素组）从一个地方移动到另一个地方。下面就来移动我们的沙滩球。

要移动沙滩球，就要改变它的位置。首先，先试着左右移动。为了确保能看到它的运动，下面把它向右移动 100 像素。在指定位置的一对数中，第一个数对应左右方向（水平方向），所以要向右移动 100 像素，需要把第一个数增加 100。我们还要加入一个延迟，以便看到动画发生。

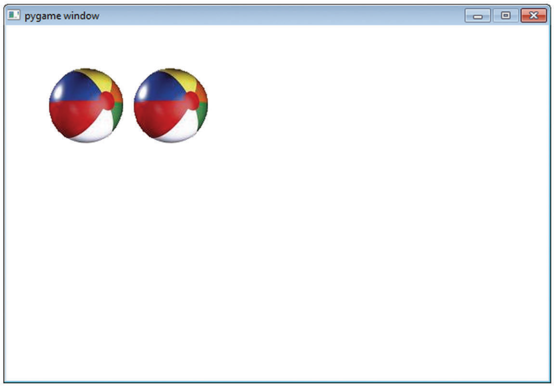
修改代码清单 16-10 的程序，改为代码清单 16-11（需要在 while 循环前增加第 8、9 和 10 行）。

代码清单 16-11 移动沙滩球

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
my_ball = pygame.image.load('beach_ball.png')
screen.blit(my_ball,[50, 50])
pygame.display.flip()
pygame.time.delay(2000)
screen.blit(my_ball,[150, 50])
pygame.display.flip()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

这是 3 行新代码

运行这个程序，看看会发生什么。球移动了吗？嗯，移动了一点。你应该会看到两个沙滩球，如右图所示。



第一个球仍然显示在原来的位置上，然后几秒之后第二个沙滩球出现在右侧。这说明我们确实把沙滩球移到了右边，但却忘了一件事：还要把第一个球擦掉！

16.7 动画

利用计算机图形做动画时，移动一个东西要完成两个步骤。

(1) 在新的位置上画出图形。
(2) 把原来的图形擦掉。

我们已经看到了第一部分。在新的位置画出了球。现在必须将原先位置的球擦掉。不过“擦掉”到底是什么意思？

擦掉图像

如果在纸上或黑板上画画，可以很容易地擦掉，只需要一块橡皮或一个黑板擦，对吗？不过，如果画的是一幅水彩画呢？假设你在画一幅蓝天的水彩画，然后在蓝天里画上一只鸟。你怎么把这只鸟“擦掉”呢？水彩是擦不掉的。你必须在鸟所在的位置上用水彩画上新的蓝天。

计算机图形就像水彩画，而不像铅笔画或粉笔画。要“擦掉”某个东西，你实际要做的是把它“盖住”。但是用什么来盖住呢？对于蓝天水彩画，天是蓝的，所以要用蓝色来覆盖小鸟。我们的背景是白色的，所以必须用白色覆盖沙滩球原来的图像。

让我们来试试看。按照代码清单 16-12 修改代码清单 16-11 中的程序。这里只需要增加一行新代码。

代码清单 16-12　再来移动沙滩球

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
```

```
screen.fill([255, 255, 255])
my_ball = pygame.image.load('beach_ball.png')
screen.blit(my_ball,[50, 50])
pygame.display.flip()
pygame.time.delay(2000)
screen.blit(my_ball, [150, 50])
pygame.draw.rect(screen, [255,255,255], [50, 50, 90, 90], 0)
pygame.display.flip()
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

这一行“擦掉”第一个球

我们增加了第 10 行，在第一个沙滩球上画了一个白色矩形。沙滩球图形大约 90 像素宽 90 像素高，所以白色矩形的大小就是 90 像素宽、90 像素高。如果运行代码清单 16-12 中的程序，看起来沙滩球会从它原来的位置移到新位置。

底下有什么

用我们的白色背景（或水彩画中的蓝天）覆盖是很容易的。不过如果在一个有云的天空里画了一只鸟，又怎么办呢？或者如果背景上有树呢？这种情况下，就必须用云或树覆盖鸟来把它擦掉。这里的重点是：你必须知道背景上有什么，也就是在你的图像“底下”是什么，因为移动图像时，必须放回或者重绘这个位置上原来的背景。

对于我们的沙滩球例子来说，这相当容易，因为背景只有白色。不过如果背景是一个沙滩场景，就会困难得多。不只是涂上白色，我们必须画出正确的背景图像部分。还有一个选择是重绘整个场景，然后把沙滩球放在它的新位置上。

16.8 更流畅的动画

目前为止，我们已经让球移动了一次！下面来看能不能用一种更逼真的方式让它移动。在屏幕上完成动画时，最好按小步移动，这样运动看起来是流畅的。下面试试用更小的步移动沙滩球。

我们并不只是让每一步更小，还要增加一个循环来移动沙滩球（因为我们希望建立很多小步）。在代码清单 16-12 的基础上编辑代码，改为代码清单 16-13 所示的程序。

如果运行这个程序，应该能看到沙滩球从原先的位置一直移动到窗口的右边。

代码清单 16-13 流畅地移动沙滩球图像

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
my_ball = pygame.image.load('beach_ball.png')
x = 50                                  增加这几行代码          使用 x 和 y（而
y = 50                                                          不是数字）
screen.blit(my_ball,[x, y])
pygame.display.flip()                开始一个 for 循环
for looper in range (1, 100):                               把 time.delay 值从 2000
    pygame.time.delay(20)                                   改变为 20
    pygame.draw.rect(screen, [255,255,255], [x, y, 90, 90], 0)
    x = x + 5
    screen.blit(my_ball, [x, y])
    pygame.display.flip()

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

让球一直移动

在前面的程序中，球一直移动到窗口右边，然后停下来。现在我们来让球一直移动下去。

如果只是增加 x 会发生什么？随着 x 值的增加，沙滩球会一直右移。不过我们的窗口（显示表面）在 x = 640 时就到头了。所以球会消失。试着把代码清单 16-13 第 10 行中的 for 循环改为：

```
for looper in range (1, 200):
```

现在循环运行次数是原先的两倍，球会从边界消失！如果希望继续看到球，有两个选择。

- 让球从窗口边界反弹。
- 让球重新翻转到窗口的另一边。

下面来看如何实现这两种做法。

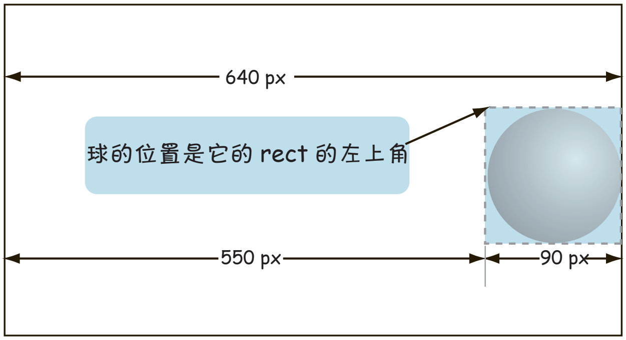
16.9 让球反弹

如果想让球看起来会在窗口的边界反弹，就要知道它什么时候“碰到”窗口边界，然后让它朝反方向移动。如果想让球一直来回移动，就要在窗口左右两边都做同样的处理。

在左边界，这很容易，因为我们只需要检查球的位置是不是等于 0（或者某个很小的数）。

在右边界，就要查看球的右边界是不是在窗口的右边界上。不过，球的位置是按它的左边界（左上角）而不是右边界设置的。所以必须减去球的宽度：

球向窗口右边移动时，位置达到 550 时要将它反弹（让它朝反方向移动）。



为了让事情变得简单一些，我们要对代码做一些修改。

- 我们希望球永远来回反弹（或者直到我们关闭 Pygame 窗口）。因为已经有了一个 while 循环，只要窗口打开，这个 while 循环就一直运行，所以我们要把显示球的代码移到这个循环内部（这就是程序最后一部分中的 while 循环）。
- 并不总是将球的位置增加 5，我们会建立一个新变量 speed，用来确定每次迭代时以多快的速度移动球。我还打算把这个值设置为 10，让球稍稍加快速度。

新代码见代码清单 16-14。

代码清单 16-14　让沙滩球反弹

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
my_ball = pygame.image.load('beach_ball.png')
x = 50
y = 50
x_speed = 10

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    pygame.time.delay(20)
    pygame.draw.rect(screen, [255,255,255], [x, y, 90, 90], 0)
    x = x + x_speed
    if x > screen.get_width() - 90  or  x < 0:
        x_speed = - x_speed
    screen.blit(my_ball, [x, y])
    pygame.display.flip()
pygame.quit()
```

这是 speed 变量

把显示球的代码放在这里，也就是 while 循环内部

当球碰到窗口的任意一边…

改变速度的符号（从正变成负，或者从负变成正），使方向反转

让球在窗口两边反弹的关键是第 19 行和第 20 行。通过第 19 行的代码（if x> screen.get_width()-90orx<0:），检查球是否在窗口边界上，如果是，就在第 20 行让它的方向反转（x_speed=-x_speed）。

试试看效果怎么样。

在 2D 空间中反弹

到目前为止，我们只是让球来回移动，或者说这只是一个方向上的运动。现在，让它同时还会上下移动。为达到这个目的，只需要再做一些修改，如代码清单 16-15 所示。

代码清单 16-15　在 2D 空间中让沙滩球反弹

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
my_ball = pygame.image.load('beach_ball.png')
x = 50
y = 50
x_speed = 10
y_speed = 10
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    pygame.time.delay(20)
    pygame.draw.rect(screen, [255,255,255], [x, y, 90, 90], 0)
    x = x + x_speed
    y = y + y_speed
    if x > screen.get_width() - 90  or  x < 0:
        x_speed = - x_speed
    if y > screen.get_height() - 90 or y < 0:
        y_speed = -y_speed
    screen.blit(my_ball, [x, y])
    pygame.display.flip()
pygame.quit()
```

为 y-speed 增加代码（垂直运动）（指向 y_speed = 10）

为 y-speed 增加代码（垂直运动）（指向 y = y + y_speed）

让球在窗口的顶边或底边反弹（指向 if y > screen.get_height() - 90 or y < 0: 及下一行）

我们在前面的程序中增加了第 9 行（y_speed=10）、第 18 行（y=y+y_speed）、第 21 行（ify> screen.get_height()-90ory<0:）和第 22 行（y_speed=-y_speed）。现在试试看效果怎么样！

如果想让球慢下来，有几种方法可以做到。

- ❒ 可以减少速度变量（x_speed 和 y_speed）。这会减少每一个动画步中球移动的距离，所以运动也会更流畅。

- 还可以增加延迟设置。在代码清单 16-15 中，延迟是 20。这是以毫秒为单位，也就是千分之一秒。所以每次循环时，程序会等待 0.02 秒。如果增加这个数值，运动会变慢。如果减少这个数值，运动就会加速。

可以试着改变速度和延迟来看最后的效果。

16.10 让球翻转

现在来看让球一直移动的第二种选择。不是让它在屏幕边界反弹，而是让它翻转。这表示，球在屏幕右边界消失时，又会在左边界重新出现。

为了让问题更简单一些，我们先来看只是水平移动球的情况。程序见代码清单 16-16。

代码清单 16-16 利用翻转移动沙滩球图像

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
my_ball = pygame.image.load('beach_ball.png')
x = 50
y = 50
x_speed = 5
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    pygame.time.delay(20)
    pygame.draw.rect(screen, [255,255,255], [x, y, 90, 90], 0)
    x = x + x_speed
    if x > screen.get_width():
        x = 0
    screen.blit(my_ball, [x, y])
    pygame.display.flip()
pygame.quit()
```

如果球在最右边……

……重新从左边开始

第 17 行（ifx>screen.get_width():）和第 18 行（x=0）中，我们检查球什么时候达到窗口的右边界，并把它移回（或者翻转）到左边界。

你可能注意到了，球在右边出现时，它会“突然跳到”[0, 50]。可能更自然的做法是从屏幕后面“滑入”。把第 18 行（x=0）改为 x=-90，再看看是不是有差别。

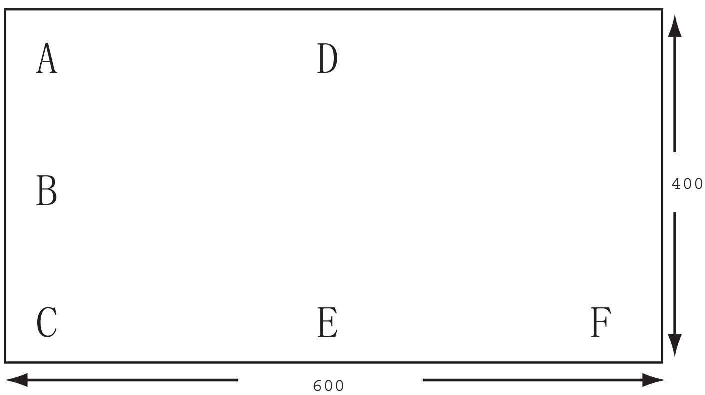
你学到了什么

哇！这一章内容真多！在这里，你学习了以下内容。

- 如何使用 Pygame。
- 如何创建图形窗口并在其中画一些形状。
- 如何设置计算机图片中的颜色。
- 如何把图像复制到图形窗口。
- 如何完成图像动画，包括将图像移动到新位置时还要从原位置“擦掉”。
- 如何让沙滩球在窗口中“反弹”。
- 如何让沙滩球在窗口中“翻转”。

测试题

1. RGB 值 [255, 255, 255] 会得到什么颜色？
2. RGB 值 [0, 255, 0] 会得到什么颜色？
3. 使用哪个 Pygame 方法来画矩形？
4. 使用哪个 Pygame 方法来画线将多个点连接在一起？
5.“像素”是什么意思？
6. 在 Pygame 窗口中，位置 [0, 0] 在哪里？
7. 如果 Pygame 窗口宽为 600 像素，高为 400 像素，下图中哪个字母位于位置 [50, 200] ？



8. 图中哪个字母位于位置 [300, 50] ？
9. 使用哪个 Pygame 方法可以将图像复制到表面（如显示表面）？
10.“移动”一个图像或完成动画时有哪两个主要步骤？

动手试一试

1. 我们讨论了画圆和矩形。Pygame 还提供了一些方法来画直线、弧、椭圆和多边形。试着在程序中使用这些方法画一些其他形状。

   可以在 Pygame 文档（www.pygame.org/docs/ref/draw.html）中了解这些方法的更多信息。如果你不能上网，在你的硬盘上也可以找到这个文档（已经随 Pygame 安装），但可能很难找到。可以搜索硬盘寻找一个名为 pygame_draw.html 的文件。

   也可以使用 Python 的帮助系统（我们在第 6 章的最后讨论过）。有一点是 SPE 做不到的，它没有提供一个交互 shell，所以要启动 IDLE，键入下面的命令：

   ```
   >>> import pygame
   >>> help()
   help> pygame.draw
   ```

   你会得到一个列表，其中会列出不同的绘制方法以及每种方法的一些解释。

2. 试着修改使用沙滩球图像的示例程序，来使用不同的图像。可以在 \examples\images 文件夹中找到一些示例图像，或者可以下载或自己画图像，还可以使用数码照片。

3. 试着改变代码清单 16-15 或代码清单 16-16 中的 `x_speed` 和 `y_speed` 值，让球移动得更快或更慢，并在不同方向上移动。

4. 试着修改代码清单 16-15，让球在隐形的墙或地板（不是窗口边界）上反弹。

5. 在代码清单 16-5 到代码清单 16-9 中（现代艺术、正弦曲线和神秘图片程序），试着把 `pygame.display.flip` 代码行移到 `while` 循环中。为此，只需加 4 个空格缩进。在这行代码后面（仍然在 `while` 循环内部），用下面这行代码增加一个延迟，看看会发生什么：

```
pygame.time.delay(30)
```

第 17 章 动画精灵和碰撞检测

在这一章中，我们将继续使用 Pygame 完成动画。这里会介绍一种动画精灵，它们能帮助我们跟踪屏幕上移动的大量图像。我们还会了解如何检测两个图像相互重叠或碰撞，比如球碰到球拍或者飞船碰到小行星。

17.1 动画精灵

从上一章我们已经了解到，看似简单的动画实际上并不简单。如果有大量图像在四处移动，要想跟踪每个图像“底下”有些什么，以便在移动图像时能够重绘，这可能要费很大的功夫。在我们的第一个沙滩球例子中，由于背景是白色的，所以更容易一些。不过你也可以想象，倘若背景上有一些图形，这肯定会复杂得多。

幸运的是，Pygame 可以为我们提供额外的帮助。四处移动的单个图像或图像部分称为动画精灵（sprite），Pygame 有一个特殊的模块来处理动画精灵。利用这个模块，我们可以更容易地移动图形对象。

在上一章中，我们让一个沙滩球在屏幕上反弹。如果希望一堆沙滩球都反弹呢？当然可以编写代码来单独地管理各个球，不过我们不会去这样做，而是使用 Pygame 的 `sprite` 模块，这样更简单一些。

术语箱

动画精灵表示作为一个单位来移动和显示的一组像素，这是一种图形对象。

"'动画精灵'（sprite）这个词是从老式的计算机和游戏机流传下来的。这些老式的游戏机不能很快地绘制和擦除图形来保证游戏正常工作。这些游戏机有一些特殊的硬件，专门用来处理需要快速移动的游戏类对象。这些对象就称为'动画精灵'。它们有一些特殊的限制，不过可以非常快地绘制和更新……如今，一般来讲，计算机的速度已经足够快了，不需要专门的硬件也可以很好地处理类似动画精灵的对象。不过'动画精灵'这个词仍用来表示二维（2D）游戏中的所有动画对象。"

（摘自 Pete Shinners 的"Pygame 教程——Sprite 模块介绍"，http://www.pygame.org/docs/tut/SpriteIntro.html。）

什么是动画精灵

可以把动画精灵想成一个小图片——一种可以在屏幕上移动的图形对象，并且可以与其他图形对象交互。

大多数动画精灵都有以下两个基本属性。

- 图像（image）：为动画精灵显示的图片。
- 矩形区（rect）：包含动画精灵的矩形区域。

动画精灵的图像可以是使用 Pygame 绘制函数绘制的图像（如上一章看到的图像），也可以是原来就有的图像文件。

Sprite 类

Pygame 的 sprite 模块提供了一个动画精灵基类，名为 Sprite。（还记得几章前讨论过的对象和类吗？）正常情况下，我们不会直接使用基类，而是基于 pygame.sprite.Sprite 来创建自己的子类。下面将完成这样一个例子，并把我们的类命名为 MyBallClass。创建这个类的代码如下：

```
class MyBallClass(pygame.sprite.Sprite):
    def __init__(self, image_file, location):
        pygame.sprite.Sprite.__init__(self)          # 初始化动画精灵
        self.image = pygame.image.load(image_file)   # 向其中加载图像文件
        self.rect = self.image.get_rect()            # 得到定义图像边界的矩形
        self.rect.left, self.rect.top = location     # 设置球的初始位置
```

要仔细分析这些代码的最后一行。location 是一个 [x, y] 位置，这是个包含两个元素的列表。因为 = 号的一边是包含两个元素的列表（x 和 y），所以可以在另一边赋两个值。这里我们为动画精灵矩形的 left 和 top 属性赋值。

既然已经定义了 MyBallClass，接下来必须创建它的一些实例。（要记住，类定义只是一个蓝图；现在必须动手盖房子。）我们仍然需要和上一章同样的代码来创建 Pygame 窗口，另外还要在屏幕上创建一些球，按行列摆放。这要利用一个嵌套循环来完成：

```
img_file = "beach_ball.png"
balls = []
for row in range (0, 3):
    for column in range (0, 3):
        location = [column * 180 + 10, row * 180 + 10]   # 每次循环时都有一个不同的位置
        ball = MyBallClass(img_file, location)            # 在这个位置创建一个球
        balls.append(ball)                                # 把球收集到一个列表中
```

我们还需要把球块移到显示表面。（还记得那个好玩的词“块移”吗？我们在上一章讨论过的。）

```
for ball in balls:
    screen.blit(ball.image, ball.rect)
pygame.display.flip()
```

把所有这些代码放在一起，就构成了我们的程序，见代码清单 17-1。

代码清单 17-1　使用动画精灵在屏幕上放多个沙滩球图像

```
import sys, pygame

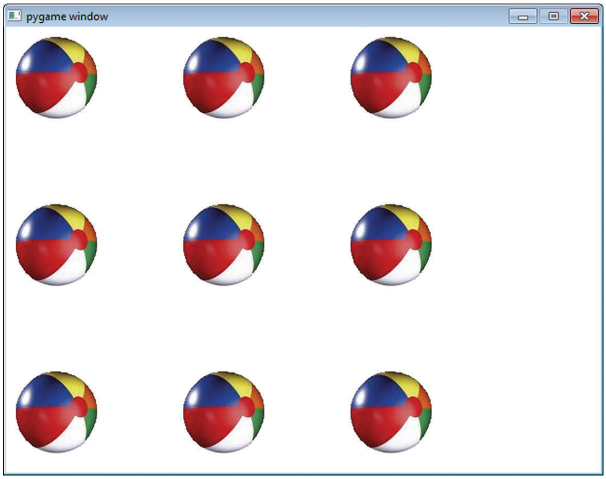
class MyBallClass(pygame.sprite.Sprite):                    # 定义 Ball 子类
    def __init__(self, image_file, location):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location

size = width, height = 640, 480                              # 设置窗口大小
screen = pygame.display.set_mode(size)
screen.fill([255, 255, 255])
img_file = "beach_ball.png"
balls = []
for row in range (0, 3):
    for column in range (0, 3):
        location = [column * 180 + 10, row * 180 + 10]
        ball = MyBallClass(img_file, location)
        balls.append(ball)                                   # 将球增加到列表
```

```
for ball in balls:
    screen.blit(ball.image, ball.rect)
pygame.display.flip()

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

如果运行这个程序，会看到 9 个沙滩球出现在 Pygame 窗口中，就像右图显示的这样：



稍后，我们就会让它们动起来。

有没有注意到第 10 行和第 11 行有一个小小的变化？（就是设置 Pygame 窗口大小的那两行代码。）我们将代码

```
screen = pygame.display.set_mode([640,480])
```

替换为

```
size = width, height = 640, 480
screen = pygame.display.set_mode(size)
```

这个代码不仅设置了窗口的大小（像前面一样），还定义了两个变量，`width` 和 `height`，后面将会用到这两个变量。这里有一点很棒，我们不仅定义了一个列表 `size`（其中包含两个元素），还定义了两个整型变量 `width` 和 `height`，而且所有这些都在一个语句中完成。另外，我们的列表两边没有加中括号，在 Python 中这是允许的。

我这样做只是想告诉你：在 Python 中有时做同样的事情可以有多种不同的方法。这些方法不存在绝对的优劣（只要它们都能工作），不能说哪种方法一定比另一种方

法强。你得遵循 Python 的语法（语言规则），这是必须保证的，尽管如此，自由表述的空间还是有的。如果你让 10 个程序员编写同样的程序，可能得不到两个完全相同的代码。

move() 方法

因为我们把球创建为 MyBallClass 的实例，应该可以使用一个类方法来移动这些球。下面就来创建一个新的类方法，名为 move()：

```
def move(self):
    self.rect = self.rect.move(self.speed)
    if self.rect.left < 0 or self.rect.right > width:
        self.speed[0] = -self.speed[0]

    if self.rect.top < 0 or self.rect.bottom > height:
        self.speed[1] = -self.speed[1]
```

检查是否碰到窗口左右两边，如果是，让 x-speed 反向

检查是否碰到窗口上下两边，如果是，让 y-speed 反向

动画精灵（实际上是其中的 rect）有一个内置方法 move()。这个方法需要一个 speed 参数来告诉它对象要移动多远（也就是移动多快）。因为我们处理的是二维（2D）图形，而 speed 是一个包含两个数的列表，一个对应 x-speed，另一个对应 y-speed。我们还要检查球是否碰到窗口的边界，使球能够在屏幕上“反弹”。

下面修改 MyBallClass 定义，增加 speed 属性和 move() 方法：

```
class MyBallClass(pygame.sprite.Sprite):
    def __init__(self, image_file, location, speed):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
        self.speed = speed

    def move(self):
        self.rect = self.rect.move(self.speed)
        if self.rect.left < 0 or self.rect.right > width:
            self.speed[0] = -self.speed[0]

        if self.rect.top < 0 or self.rect.bottom > height:
            self.speed[1] = -self.speed[1]
```

增加 location 参数

增加这行代码，为球创建一个 speed 属性

增加这个方法来移动球

注意第 2 行（def__init__(self,image_file,location,speed):）中的修改，这里增加了第 7 行（self.speed=speed），另外第 9 行到第 15 行增加了新的 move() 方法。

现在创建球的各个实例时，需要告诉它速度，还要指出图像文件以及位置：

```
speed = [2, 2]
ball = MyBallClass(img_file, location, speed)
```

前面的代码把所有球都创建为相同的速度（相同方向），不过如果球的移动有些随机性可能更有意思。下面使用 random.choice() 函数来设置速度，如下：

```
from random import *
speed = [choice([-2, 2]), choice([-2, 2])]
```

这会为 x 和 y 速度选择 –2 或 2。

代码清单 17-2 给出了完整的程序。

代码清单 17–2　使用动画精灵移动球的程序

```
import sys, pygame
from random import *

class MyBallClass(pygame.sprite.Sprite):
    def __init__(self, image_file, location, speed):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
        self.speed = speed

    def move(self):
        self.rect = self.rect.move(self.speed)
        if self.rect.left < 0 or self.rect.right > width:
            self.speed[0] = -self.speed[0]

        if self.rect.top < 0 or self.rect.bottom > height:
            self.speed[1] = -self.speed[1]

size = width, height = 640, 480
screen = pygame.display.set_mode(size)
screen.fill([255, 255, 255])
img_file = "beach_ball.png"
balls = []
for row in range (0, 3):
    for column in range (0, 3):
        location = [column * 180 + 10, row * 180 + 10]
        speed = [choice([-2, 2]), choice([-2, 2])]
        ball = MyBallClass(img_file, location, speed)
        balls.append(ball)
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    pygame.time.delay(20)
```

Ball 类定义

创建列表跟踪所有球

创建各个球时把球增加到列表

```
    screen.fill([255, 255, 255])
    for ball in balls:
            ball.move()
            screen.blit(ball.image, ball.rect)
    pygame.display.flip()
pygame.quit()
```

重绘屏幕

这个程序使用一个列表来跟踪所有球。第 32 行（balls.append(ball)）上，创建每个球时会把球增加到这个列表。

最后 5 行代码会重绘屏幕。这里我们走了一条捷径，并不是单独地“擦除”（覆盖）各个球，我们直接用白色填充窗口，然后重绘所有球。

可以试验一下这些代码，看看有更多（或更少）球时会怎么样，可以改变它们的速度，还可以改变它们移动和“反弹”的方式，等等。你会注意到，球会四处移动，而且在窗口四周会反弹，不过它们相互之间还不能反弹！

17.2 嘣！碰撞检测

大多数计算机游戏中，你需要知道一个动画精灵在什么时候碰到另一个精灵。例如，可能需要知道保龄球何时碰到球瓶，或者导弹什么时候击中飞船。

你可能认为，如果我们知道每个动画精灵的位置和大小，可以写一些代码对每一个其他动画精灵的位置和大小进行检查，看哪里出现了重叠。不过，编写 Pygame 的人已经为我们完成了这项工作。Pygame 中已经内置有这种碰撞检测。

术语箱

简单地说，碰撞检测（collision detection）指的是了解两个动画精灵何时接触或重叠。两个移动的东西相互碰到一起，这就是一个碰撞（collision）。

Pygame 还提供了一种方法对动画精灵分组。例如，在保龄球游戏中，所有球瓶可能在一组，球则在另一组。

组和碰撞检测密切相关。在保龄球例子中，你可能想检测球何时击倒某个瓶子，因此要寻找球精灵与球瓶组中所有精灵之间的碰撞。还可以检测组内部的碰撞（如球瓶相互碰倒）。

下面来完成一个例子。以我们的反弹沙滩球为基础，不过为了更容易地看出发生了什么，这里首先建立 4 个球而不是 9 个球。另外与上一个例子中建立球的列表不同，我们将会使用 Pygame 的 group 类。

这里还要对代码稍稍做些整理，把完成球动画的部分（代码清单 17-2 中的最后几行）放在一个函数中，我们把这个函数命名为 animate()。animate() 函数还包括完成碰撞检测的代码。两个球碰撞时，我们会让它们反向。

代码清单 17-3 显示了相应的代码。

代码清单 17-3　使用一个动画精灵组而不是列表

```
import sys, pygame
from random import *

class MyBallClass(pygame.sprite.Sprite):
    def __init__(self, image_file, location, speed):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
        self.speed = speed

    def move(self):
        self.rect = self.rect.move(self.speed)
        if self.rect.left < 0 or self.rect.right > width:
            self.speed[0] = -self.speed[0]
        if self.rect.top < 0 or self.rect.bottom > height:
            self.speed[1] = -self.speed[1]

def animate(group):
    screen.fill([255,255,255])
    for ball in group:
        group.remove(ball)
        if pygame.sprite.spritecollide(ball, group, False):
            ball.speed[0] = -ball.speed[0]
            ball.speed[1] = -ball.speed[1]

        group.add(ball)
        ball.move()
        screen.blit(ball.image, ball.rect)
    pygame.display.flip()
    pygame.time.delay(20)
size = width, height = 640, 480
screen = pygame.display.set_mode(size)
screen.fill([255, 255, 255])
img_file = "beach_ball.png"
group = pygame.sprite.Group()
for row in range (0, 2):
    for column in range (0, 2):
        location = [column * 180 + 10, row * 180 + 10]
        speed = [choice([-2, 2]), choice([-2, 2])]
        ball = MyBallClass(img_file, location, speed)
        group.add(ball)

running = True
while running:
```

（代码旁注释：Ball 类定义；新的 animate() 函数；从组删除精灵；检查精灵与组之间的碰撞；将球再添加回原来的组中；主程序从这里开始；创建精灵组；这一次只创建 4 个球；将各个球增加到组）

```
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    animate(group)
pygame.quit()
```

调用 animate() 函数并传入 group

这里最有意思的新内容是碰撞检测如何工作。Pygame sprite 模块有一个 spritecollide() 函数，它会查找一个精灵与一个组中所有精灵之间的碰撞。要检查同一个组中精灵之间的碰撞，必须通过 3 步来完成：

- 从这个组中删除这个精灵；
- 检查这个精灵与组中其他精灵之间的碰撞；
- 再把这个精灵添加回原来的组中。

这些工作在第 21 行到第 29 行的 for 循环中（animate() 函数的中间部分）完成。如果开始时没有从组中删除这个精灵，spritecollide() 会检测到这个精灵与它自身发生了碰撞，因为它也在这个组中。乍一看好像有些奇怪，不过如果再想想看就会发现这是有道理的。

运行程序，看看有什么结果。有没有注意到一些奇怪的行为？我注意到两点：

- 球碰撞时，它们会“颤抖”或者发生两次碰撞；
- 有时球会“卡”在窗口边界上，颤抖一段时间。

> 如果加大动画步，就能更容易地看到这一点。可以把速度从 2 增加到 5，另外把各步之间的延迟从 20 增加到 50。

为什么会出现这种情况？嗯，这与我们如何编写 animate() 函数有关。注意我们的做法是先移动一个球，检查它的碰撞，然后移动另一个球，再检查这个球的碰撞，依此类推。也许应该先完成所有移动，然后再完成全部碰撞检测。

所以需要把第 28 行（ball.move()）放在它自己的循环中，就像这样：

```
def animate(group):
    screen.fill([255,255,255])
    for ball in group:
        ball.move()
    for ball in group:
        group.remove(ball)

        if pygame.sprite.spritecollide(ball, group, False):
            ball.speed[0] = -ball.speed[0]
            ball.speed[1] = -ball.speed[1]

        group.add(ball)
```

先移动所有球

再完成碰撞检测实现反弹

```
        screen.blit(ball.image, ball.rect)
    pygame.display.flip()
    pygame.time.delay(20)
```

试试看，效果是不是比原来好一些。

可以对这个代码做些试验，改变某些值，比如速度（time.delay() 数）、球数、球原先的位置、随机性等，来看球会有什么变化。

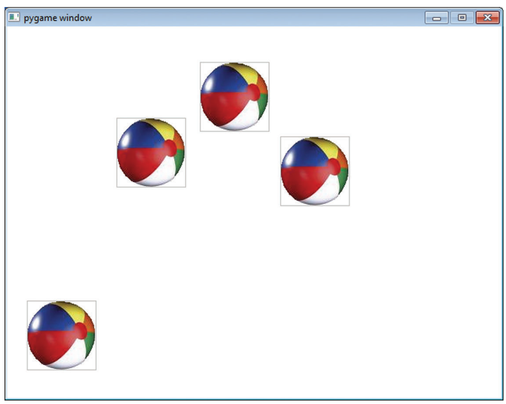
矩形碰撞与像素完美碰撞

你会注意到，球“碰撞”时并不总是完全接触。这是因为 spritecollide() 没有使用球的圆形轮廓来检测碰撞。它使用了球的 rect，也就是球的外围矩形。

如果想看看具体是怎样的，可以画一个矩形包围球图像，并且使用这个新图像而不是原先常规的沙滩球图像。我已经为你做好了这个新图像，你可以试一试：

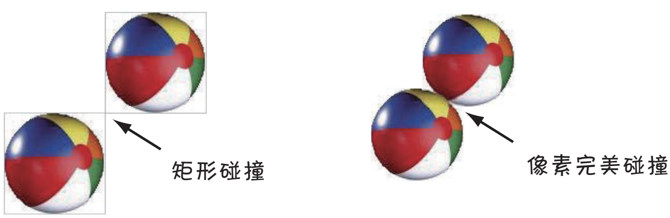
```
img_file = "b_ball_rect.png"
```

看上去就像右图显示的这样：



如果希望球的圆形部分（而不是矩形边界）真正接触时球才会相互反弹，就必须使用一种称为“像素完美碰撞检测”的方法。spritecollide() 函数没有这样做，而是使用了更简单的“矩形碰撞检测”。

它们的区别如下。使用矩形碰撞检测，两个球矩形区的任何部分相互接触时就会“碰撞”。而使用像素完美碰撞检测，两个球本身接触时才会碰撞。如下：

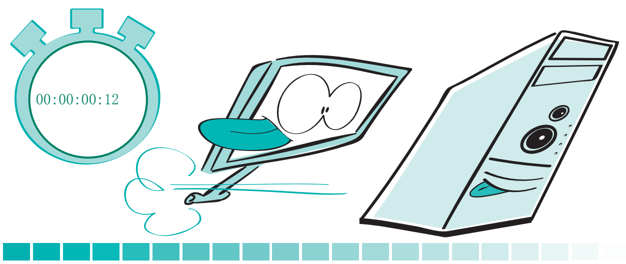


像素完美碰撞检测更逼真。（在真正的沙滩球周围你不会觉得有任何隐形的矩形，对吧？）但是在程序中实现时就没这么简单了。

对于要在 Pygame 中完成的大多数工作来说，矩形碰撞检测已经足够了。像素完美碰撞检测需要写更多代码，而且会让游戏运行得更慢，所以应该只有在确实有必要的情况下才使用这种方法。完成像素完美碰撞检测有几个模块（我上次查看时，Pygame 网站上至少有两个）。如果想尝试进行像素完美碰撞检测，只需在网上搜索一下就会找到这些模块。

17.3 统计时间

到目前为止，我们一直在使用 `time.delay()` 来控制动画运行的快慢。不过这不是最好的办法，这是因为，使用 `time.delay()` 时，你并不真正知道每个循环需要多长时间。循环中的代码要花一些时间来运行（这是一个未知时间），然后延迟也要花费一些时间（这是一个已知时间）。所以这个时间中有一部分是已知的，但有一部分是未知的。



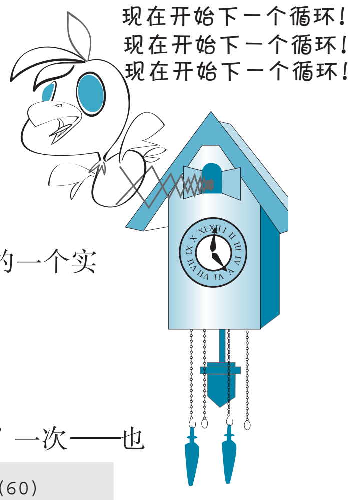
如果我们想知道循环多长时间运行一次，就需要知道每个循环的总时间，这应当是代码运行时间 + 延迟时间。要计算动画的时间，使用毫秒或千分之一秒会很方便。它的缩写是 ms，所以 25 毫秒就是 25 ms。

在我们的例子中，假设代码时间是 15 ms。这说明，`while` 循环中的代码运行需要 15 ms，这不包括 `time.delay()`。我们已经知道延迟时间，因为这里使用 `time.delay(20)` 把延迟设置为 20 ms。所以循环的总时间是 20 ms + 15 ms = 35 ms。由于 1 秒就是 1000 ms，如果每个循环需要 35 ms，可以得到 1000 ms / 35 ms = 28.57。这说明每秒大约有 29 个循环。在计算机图形学中，每个动画步叫做一帧，游戏程序员讨论图形更新的快慢时都会提到帧速率（每秒帧数，fps）。在我们的例子中，帧速率大约是 29 fps。

问题在于，我们并不能真正控制这个公式中的“代码时间”部分。如果增加或删除代码，这个时间就会改变。即使是相同的代码，如果动画精灵个数不同（例如，随着游戏对象的出现和消失，动画精灵个数会变化），绘制这些精灵所花费的时间也

会变化。此外，在不同的机器上，相同的代码运行的速度也不同。可能不是 15 ms，代码时间可能变成 10 ms 或 20 ms。如果有一种更便于预测的方法来控制帧速率就好了。好在，Pygame 的 time 模块为我们提供了这样的工具：一个名为 Clock 的类。

用 pygame.time.Clock() 控制帧速率



并不是向每个循环增加一个延迟，pygame.time.Clock() 会控制每个循环多长时间运行一次。这就像一个定时器在控制时间进程，指出“现在开始下一个循环！现在开始下一个循环！……”

使用 Pygame 时钟之前，必须先创建 Clock 对象的一个实例。这与创建其他类的实例完全相同：

```
clock = pygame.time.Clock()
```

然后在主循环体中，只需要告诉时钟多久“滴答”一次——也就是说，循环应该多长时间运行一次：

```
clock.tick(60)
```

传入 clock.tick() 的数不是一个毫秒数。这是每秒内循环要运行的次数。所以这个循环应当每秒运行 60 次。在这里我只是说“应当运行”，因为循环只能按计算机能够保证的速度运行。每秒 60 个循环（或帧）时，每个循环需要 1000 / 60 = 16.66 ms（大约 17 ms）。如果循环中的代码运行时间超过 17 ms，在 clock 指出开始下一次循环时当前循环将无法完成。

实际上，这说明对于图形运行的帧速率有一个限制。这个限制取决于图形的复杂程度、窗口大小以及运行这个程序的计算机的速度。对于一个特定的程序，计算机的运行速度可能是 90 fps，而较早的一个较慢的计算机也许只能以 10 fps 的速度缓慢运行。

对于非常复杂的图形，大多数现代计算机都完全可以按 20 ～ 30 fps 的速率运行 Pygame 程序。所以如果希望你的游戏在大多数计算机上都能以相同的速度运行，可以选择一个 20 ～ 30 fps（或者更低）的帧速率。这已经很快了，足以生成看上去流畅的运动。从现在开始，这本书中的例子都将使用 clock.tick(30)。

检查帧速率

如果想知道你的程序能以多快的速度运行，可以用一个名为 clock.get_fps()

的函数检查帧速率。当然，如果将帧速率设置为 30，它就总会以 30 fps 的帧速率运行（假设你的计算机能够运行那么快）。要看一个特定程序在特定机器上运行的最快速度，可以先将 clock.tick 设置得非常快（例如 200 fps），然后运行这个程序，用 clock.get_fps() 检查实际的帧速率。（接下来就会给出一个这样的例子。）

调整帧速率

如果想要确保你的动画在每个机器上都以相同的速度运行，可以利用 clock.tick() 和 clock.get_fps() 实现一个小技巧。因为你知道要以多快的速度运行，而且也知道实际运行的速度，因此可以根据机器的速度调整（scale）动画的速度。

例如，假设已经设置了 clock.tick(30)，这说明你想按 30 fps 的帧速率运行。如果使用 clock.get_fps() 并发现只得到速率为 20 fps，可以知道：屏幕上对象移动的速度比你希望的要慢。因为每秒的帧数更少，所以每一帧必须把对象移动得更远，这样看上去才跟得上预想的速度。你的移动对象可能有一个名为 speed 的变量（或属性），这会告诉它们每一帧移动多远。只需要增加 speed 对运行速度较慢的机器做出补偿。

要增加多少呢？可以按期望帧频率与实际帧速率的比值来增加。如果对象的当前速度是 10，期望的帧速率是 30 fps，程序实际运行速率为 20 fps，可以得到：

```
object_speed = current_speed * (desired fps / actual fps)
object_speed = 10 * (30 / 20)
object_speed = 15
```

所以并不是每帧要将对象移动 10 个像素，而是需要移动 15 个像素，才能弥补较慢的帧速率。我们将在本书后面的一些程序中使用这个技巧。

下面的沙滩球程序使用了前面几节讨论的内容：Clock 和 get_fps()。

代码清单 17-4 沙滩球程序中使用 Clock 和 get_fps()

```
import sys, pygame
from random import *
class MyBallClass(pygame.sprite.Sprite):
    def __init__(self, image_file, location, speed):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
        self.speed = speed
    def move(self):
        self.rect = self.rect.move(self.speed)
        if self.rect.left < 0 or self.rect.right > width:
```

Ball 类定义

```
            self.speed[0] = -self.speed[0]
        if self.rect.top < 0 or self.rect.bottom > height:
            self.speed[1] = -self.speed[1]

def animate(group):
    screen.fill([255,255,255])
    for ball in group:
        ball.move()
    for ball in group:
        group.remove(ball)
        if pygame.sprite.spritecollide(ball, group, False):
            ball.speed[0] = -ball.speed[0]
            ball.speed[1] = -ball.speed[1]
        group.add(ball)
        screen.blit(ball.image, ball.rect)
    pygame.display.flip()

size = width, height = 640, 480
screen = pygame.display.set_mode(size)
screen.fill([255, 255, 255])
img_file = "beach_ball.png"
clock = pygame.time.Clock()
group = pygame.sprite.Group()
for row in range (0, 2):
    for column in range (0, 2):
        location = [column * 180 + 10, row * 180 + 10]
        speed = [choice([-4, 4]), choice([-4, 4])]
        ball = MyBallClass(img_file, location, speed)
        group.add(ball)   #add the ball to the group

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
            frame_rate = clock.get_fps()
            print "frame rate = ", frame_rate
    animate(group)
    clock.tick(30)
pygame.quit()
```

动画函数

已经删除 time.delay()

创建 Clock 的实例

初始化并画出沙滩球

主 while 循环从这里开始

检查帧速率

clock.tick 现在控制帧速率（受计算机速度限制）

Pygame 和动画精灵的基本知识就介绍完了。在下一章中，我们将使用 Pygame 建立一个真正的游戏，我们还会介绍另外一些你能完成的工作，比如增加文本（显示游戏得分）、声音和鼠标及键盘输入。

你学到了什么

在这一章，你学到了以下内容。

- Pygame 中的动画精灵，以及如何使用动画精灵处理多个移动的图像。

- 动画精灵组。
- 碰撞检测。
- pygame.clock 和帧速率。

测试题

1. 什么是碰撞检测？
2. 什么是像素完美碰撞检测？它与矩形碰撞检测有什么区别？
3. 可以利用哪两种方法跟踪多个在一起的动画精灵对象？
4. 在代码中控制动画的速度有哪两种方法？
5. 为什么使用 pygame.clock 比使用 pygame.time.delay() 更准确？
6. 怎么得出你的程序运行的帧速率？

动手试一试

键入这一章中的所有代码示例就能让你试个够。如果还不够，可以回过头去再做一遍。相信你能从中得到很多收获！

第 18 章
一种新的输入——事件

到目前为止，我们已经向程序提供过几种非常简单的输入。用户可以使用 `raw_input()` 键入字符串，或者我们可以从 EasyGui（见第 6 章）得到数字和字符串。我们还介绍了如何使用鼠标来关闭一个 Pygame 窗口，不过我还没有解释这是怎么做到的。

这一章中，你将学习一种不同的输入，叫做事件（event）。在这里，我们会具体分析 Pygame 窗口退出代码做了些什么，以及它是如何做到的。我们还将从鼠标得到输入，另外会让程序对按键立即做出反应，而不必等待用户按下回车。

18.1 事件

如果我在现实生活中问你，“什么是事件”，你可能会说这是“发生的某件事情”。这是一个很好的定义，这个定义在编程中也同样适用。很多程序都需要对“发生的事情”做出反应。比如说：

- 移动或点击鼠标；
- 按键；
- 经过了一定时间。

目前为止，我们写的大多数程序自始至终都沿着一条可以预测的路径运行，可能中间会有一些循环或条件。不过，除此以外还有另外一类程序，称为事件驱动程序（event-driven program），它们的做法完全不同。事件驱动程序基本上只是“原地不动”，什么也不做，等待着有事件发生。一旦事件确实发生，它们就会做出反应，完成所有必要的工作来处理这个事件。

Windows 操作系统（或者其他 GUI）就是这种事件驱动程序的一个很好的例子。打开 Windows 计算机时，启动后它只是“原地不动”，不会启动任何程序，你也不会

看到鼠标光标在屏幕上移动。不过，如果你开始移动或点击鼠标，就会有情况发生。鼠标光标会在屏幕上移动，“开始”菜单会弹出，或者会做其他事情。

事件循环

为了让一个事件驱动程序“看到”有事件发生，它必须“寻找”这些事件。程序必须不断地扫描计算机内存中用来指示事件发生的部分。只要程序在运行，就会反复这样做。回顾第 8 章，我们已经了解了程序如何反复做某些事情，这要使用一个循环。不断寻找事件的这个特殊循环叫做事件循环（event loop）。

前两章完成的 Pygame 程序中，最后总是有一个 `while` 循环。我们说过，这个循环会在程序运行期间一直运行。这个 `while` 循环就是 Pygame 的事件循环。（要了解退出代码是如何工作的，首先要知道这个事件循环。）

事件队列

只要有人移动或点击了鼠标或者按下了按键，就会发生事件。这些事件去哪里了呢？在上一节中我说过，事件循环会一直不断地搜索内存的某个部分。内存中存储事件的部分叫做事件队列（event queue）。

术语箱

> 队列（queue）读作“Q”。日常生活中，这就表示排队。
>
> 在编程中，队列通常指一个列表，其中的元素按某种特定的顺序到达，或者将按某种特定的顺序使用。

事件队列就是发生的所有事件的列表，这些事件按它们发生的顺序排列。

事件处理器

如果编写一个 GUI 程序或游戏，程序必须知道用户什么时候按下一个按键或者移动了鼠标。这些按键、点击和移动鼠标都是事件，而且程序必须知道如何应对这些事件，它必须处理事件。程序中处理某个事件的部分称为一个事件处理器（event handler）。

并不是每一个事件都要处理。在桌面上移动鼠标时，会创建成百上千个事件，因为事件循环运行得非常快。每一个瞬间（远远不到 1 秒），即使鼠标只是移动了一点点，也会生成一个新的事件。不过你的程序可能并不关心鼠标的每一个小小的移动。它可能只关心用户什么时候点击某个部分。所以你的程序可以忽略 `mouseMove` 事件，只关注 `mouseClick` 事件。

事件驱动程序中，对于所关心的各种事件会有相应的事件处理器。如果你有一个游戏使用键盘上的方向键来控制一艘船的移动，可能要为 keyDown 事件编写一个处理器。相反，如果使用鼠标控制这艘船，就可能为 mouseMove 事件写一个事件处理器。

现在就来看我们的程序中可以使用的一些具体事件。我们还会使用 Pygame，所以这一章后面讨论的所有事件都来自 Pygame 的事件队列。其他 Python 模块会提供不同的事件。例如，我们将在第 20 章讨论另外一个名为 PyQt 的模块。PyQt 有自己的事件集，其中一些事件与 Pygame 有所不同。不过，对于不同的事件集（甚至在不同的编程语言中），处理事件的方式通常都是一样的。对于每个事件系统来说可能都不完全一样，不过相同点还是远远多于不同点。

18.2　键盘事件

下面先来看一个键盘事件的例子。假设我们希望一旦按下键盘上的某个键就做某件事情。在 Pygame 中，这个事件是 KEYDOWN。为了说明这个事件如何使用，下面还是用代码清单 16-15 中反弹球的例子，球会向两边移动，并在窗口边界反弹。不过在增加事件之前，下面先更新这个程序，加入我们刚学到的一些新内容：

- 使用动画精灵；
- 使用 clock.tick() 而不是 time.delay()。

首先，需要为球建立一个类。这个类要有一个 __init__() 方法和一个 move() 方法。我们将创建这个类的实例，另外在主 while 循环中将使用 clock.tick(30)。代码清单 18-1 显示了修改后的代码。

代码清单 18-1　反弹球程序，加入动画精灵和 Clock.tick()

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])
background = pygame.Surface(screen.get_size())
background.fill([255, 255, 255])
clock = pygame.time.Clock()
class Ball(pygame.sprite.Sprite):
    def __init__(self, image_file, speed, location):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
        self.speed = speed
```

Ball 类，包括 move() 方法

```
    def move(self):
        if self.rect.left <= screen.get_rect().left or \
                self.rect.right >= screen.get_rect().right:
            self.speed[0] = - self.speed[0]
        newpos = self.rect.move(self.speed)
        self.rect = newpos

my_ball = Ball('beach_ball.png', [10,0], [20, 20])     创建 Ball 的一个实例
running = True                            速度，位置
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    clock.tick(30)                        这是时钟
    screen.blit(background, (0, 0))
    my_ball.move()
    screen.blit(my_ball.image, my_ball.rect)          完全重绘
    pygame.display.flip()
pygame.quit()
```

这里要注意一个问题，移动球时我们没有“擦除”球，而是做了不同的处理。我们已经知道，在新位置上重画球之前要从原位置“擦除”动画精灵有两种方法：一种方法是在每个动画精灵的原位置上涂上背景颜色，另一种方法是直接重绘每一帧的整个背景——实际上每一次都会从一个空屏幕开始。在这里，我们采用了第二种做法。不过这里不是每次循环时使用 screen.fill()，而是建立了一个名为 background 的表面，用白色填充。每次循环时，只需把这个背景“块移”到显示表面 screen。这样也能达到目的；这只是完成这项工作的不同方法而已。

按键事件

现在我们要增加一个事件处理器，当按下向上箭头时让球上移，按下向下箭头时让球下移。Pygame 包括多个不同模块。这一章中我们将使用的模块是 pygame.event。

我们已经保证 Pygame 事件循环会一直运行（while 循环）。这个循环在扫描一个名为 QUIT 的特殊事件。

```
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
```

pygame.event.get() 方法从事件队列得到所有事件的一个列表。for 循环迭代处理这个列表中的每一个事件，如果看到一个 QUIT 事件，它会将 running 设置为 False，这会导致 while 循环结束，并结束程序。了解到这一点后，现在你应该已经完全清楚“点击 × 结束程序”代码是如何工作的。

不过对于这个例子，我们还希望检测另外一种不同类型的事件。我们希望知道

何时按下一个按键，所以要查找 KEYDOWN 事件。我们需要这样的代码：

```
if event.type == pygame.KEYDOWN
```

由于前面已经有了一个 if 语句，可以直接用 elif 增加另一个条件（我们已经在第 7 章介绍过这个内容）：

```
while running:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        elif event.type == pygame.KEYDOWN:     # 这是用来检测按键的新增部分
            # do something
```

按下按键时我们想做什么呢？我们说过，如果按下向上箭头，要让球上移，如果按下向下箭头，要让球下移。所以可以这样做：

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                my_ball.rect.top = my_ball.rect.top - 10     # 让球上移 10 个像素
            elif event.key == pygame.K_DOWN:
                my_ball.rect.top = my_ball.rect.top + 10     # 让球下移 10 个像素
```

K_UP 和 K_DOWN 是 Pygame 中向上和向下箭头的名字。对代码清单 18-1 完成以上修改，程序现在如代码清单 18-2 所示。

代码清单 18-2　响应向上和向下箭头键的反弹球

```
import pygame, sys                                      # 初始化
pygame.init()
screen = pygame.display.set_mode([640,480])
background = pygame.Surface(screen.get_size())
background.fill([255, 255, 255])
clock = pygame.time.Clock()

class Ball(pygame.sprite.Sprite):                       # Ball 类定义，包括 move() 方法
    def __init__(self, image_file, speed, location):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
        self.speed = speed

    def move(self):
```

```
        if self.rect.left <= screen.get_rect().left or \
                self.rect.right >= screen.get_rect().right:
            self.speed[0] = - self.speed[0]
        newpos = self.rect.move(self.speed)
        self.rect = newpos

my_ball = Ball('beach_ball.png', [10,0], [20, 20])      ◁ 创建 Ball 的一个实例
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                my_ball.rect.top = my_ball.rect.top - 10       检查按键，让
            elif event.key == pygame.K_DOWN:                   球上移或下移
                my_ball.rect.top = my_ball.rect.top + 10

    clock.tick(30)
    screen.blit(background, (0, 0))
    my_ball.move()                                             完全重绘
    screen.blit(my_ball.image, my_ball.rect)
    pygame.display.flip()
pygame.quit()
```

运行代码清单 18-2 中的程序，试着按下向上箭头和向下箭头。起作用吗？

重复按键

你可能已经注意到，如果保持按下向上或向下箭头不放，球只会向上或向下移动一步。这是因为，我们没有告诉程序如果按键一直按下时该怎么做。用户按键时，会生成一个 KEYDOWN 事件，不过 Pygame 中还有一个设置，可以在按键一直按下时生成多个 KEYDOWN 事件。这称为按键重复（key repeat）。你要告诉它开始重复之前等待多长时间，另外还要指出多长时间重复一次。这些值的单位都是毫秒（千分之一秒）。可能像这样：

```
delay = 100
interval = 50
pygame.key.set_repeat(delay, interval)
```

delay 值告诉 Pygame 在开始重复之前等待多长时间，interval 值告诉 Pygame 按键要以多快的速度重复，也就是说，各个 KEYDOWN 事件之间要间隔多长时间。

试着把这个代码增加到代码清单 18-2（放在 pygame.init 后面，不过要在 while 循环前面），看看这会让程序的行为有什么变化。

事件名和按键名

查找按下的向上或向下箭头时，我们要寻找 KEYDOWN 事件类型以及 K_UP 和 K_

DOWN 按键名。还有其他事件吗？其他按键名是什么？

实际上还有相当多的事件，所以这里不打算一一列出。不过 Pygame 网站上提供了所有事件的列表。可以在 Pygame 文档的 event 部分找到这个事件列表：

www.pygame.org/docs/ref/event.html

按键名列表放在 key 部分：

www.pygame.org/docs/ref/key.html

以下是我们将要使用的一些常用事件：

- QUIT
- KEYDOWN
- KEYUP
- MOUSEMOTION
- MOUSEBUTTONUP
- MOUSEBUTTONDOWN

Pygame 还给每个可以按下的键命了名。我们刚才看到了向上和向下箭头，它们的名字分别是 K-UP 和 K-DOWN。后面还会看到另外一些按键名，它们都以 K_ 开头，后面是按键的名字，例如：

- K_a，K_b（对应字母键）
- K_SPACE
- K_ESCAPE

等等。

18.3　鼠标事件

我们刚才看到了如何从键盘得到按键事件，以及如何使用这些事件来控制程序中的某些方面。前面使用箭头键让沙滩球向上和向下移动。现在打算使用鼠标来控制球。从中你会了解到如何处理鼠标事件以及如何使用鼠标位置信息。

最常用的 3 类鼠标事件如下：

- MOUSEBUTTONUP
- MOUSEBUTTONDOWN
- MOUSEMOTION

最简单的事情是：只要鼠标在 Pygame 窗口中移动，就让沙滩球随着鼠标位置移动。要移动沙滩球，我们将使用球的 rect.center 属性。这样一来，球的中心就会跟着鼠标移动。

我们要把 while 循环中检测按键事件的代码替换为检测鼠标事件。

```
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEMOTION:
            my_ball.rect.center = event.pos        检测鼠标移动并移动球
```

这比检测键盘事件还要简单。对代码清单 18-2 完成以上修改，并试着运行这个程序。event.pos 部分是鼠标的位置（x 和 y 坐标）。只需要把球的中心移动到这个位置。注意，只要鼠标在移动，球就跟着移动。也就是说，只要 MOUSEMOVE 事件正在发生，球就会随着移动。改变球的 rect.center 会同时改变 x 和 y 位置。我们不再只是让球向上或向下移动，而是会上下左右同时移动。如果没有鼠标事件（可能因为鼠标没有移动，或者鼠标光标落在 Pygame 窗口之外），球就会继续在左右两边反弹。

现在试着只是在鼠标按钮保持按下时才让鼠标控制起作用。鼠标按钮保持按下时移动鼠标称为拖动（dragging）。并没有一种 MOUSEDRAG 事件类型，所以需要使用现有的事件类型来得到我们希望的效果。

如何区分是否在拖动鼠标呢？拖动意味着鼠标移动时鼠标按钮一直保持按下。我们可以利用 MOUSEBUTTONDOWN 事件得到鼠标按钮何时按下，另外利用 MOUSEBUTTONUP 事件可以得到按钮何时松开（还原，不再按下）。因此只需跟踪按钮的状态，可以通过建立一个变量来做到，我们将这个变量命名为 held_down。具体做法如下：

```
held_down = False
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEBUTTONDOWN:
            held_down = True                        确定鼠标按钮
        elif event.type == pygame.MOUSEBUTTONUP:    是否保持按下
            held_down = False
        elif event.type == pygame.MOUSEMOTION:
            if held_down:
                my_ball.rect.center = event.pos   <-- 拖动鼠标时才执行
```

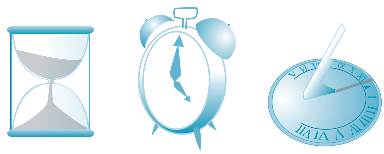
拖动条件（鼠标移动时鼠标按钮保持按下）在以上代码的最后一个 elif 块中检测。前面已经修改过代码清单 18-2，在这个修改后的代码中，对 while 循环完成上



述修改。运行这个程序，看看它的效果。

嘿，要知道我们从第 1 章就已经开始编程了！不过，因为现在开始使用图形、动画精灵和鼠标，所以变得更有意思了。前面说过会谈到这些内容。不过你要跟上我的思路，先来学习一些基础知识。

18.4　定时器事件



在这一章中，目前为止我们已经见过键盘事件和鼠标事件。另一种非常有用的事件（特别是在游戏和仿真中）是定时器事件（timer event）。定时器会按固定的间隔生成事件，就像你的闹钟一样。如果你设好闹钟，并把闹铃打开，每天它都会在固定的时刻响起来。

可以把 Pygame 定时器设置为任意间隔。如果定时器到时间，它会创建一个能够被事件循环检测到的事件。那么它会生成什么类型的事件呢？它生成的是一种用户事件（user event）。

Pygame 有很多预定义的事件类型。这些事件会编号（从 0 开始），它们还有自己的名字以便我们记住。我们已经见过一些事件名，比如 MOUSEBUTTONDOWN 和 KEYDOWN。除此以外，Pygame 还为用户定义的事件（user-defined event）留出了很大空间。这些事件不是 Pygame 为特定事件预留的，你可以用它们表示任何事情，其中之一就是定时器。

要在 Pygame 中设置定时器，要使用 set_timer() 函数，如下：

```
pygame.time.set_timer(EVENT_NUMBER, interval)
```

EVENT_NUMBER 是事件编号，interval 是定时器多长时间（单位是毫秒）到期并生成一个事件。

要使用什么 EVENT_NUMBER 呢？应当使用 Pygame 还没有用过的一个编号（也就是说，尚未将这个编号用于其他事件）。可以询问 Pygame 哪些编号已经占用。可以

在交互模式中执行下面的命令：

```
>>> import pygame
>>> pygame.USEREVENT
24
```

这会告诉我们，Pygame 正在使用从 0 到 23 的事件编号，对于用户事件，第一个可用的编号是 24。所以需要选择 24 或一个更大的数。可以大到什么程度呢？可以再来问一问 Pygame。

```
>>> pygame.NUMEVENTS
32
```

NUMEVENTS 告诉我们 Pygame 中可以有的事件类型最大编号是 32（从 0 到 31）。所以必须选择一个大于或等于 24 但小于 32 的数。可以像这样直接设置定时器：

```
pygame.time.set_timer(24, 1000)
```

不过，如果出于某种原因 USEREVENT 的值有变化，这个代码可能就无法正常工作了。可能这样做会更好一些：

```
pygame.time.set_timer(pygame.USEREVENT, 1000)
```

如果我们必须建立另一个用户事件，可以使用 USEREVENT + 1，依此类推。这个例子中的 1000 表示 1000 毫秒，也就是 1 秒，所以这个定时器每秒响一次。下面把这个定时器放入我们反弹球程序中。

像前面一样，我们将利用事件让球上移或下移，不过由于这一次球并非由用户来控制，我们要让它除了在左右两边反弹还会在上下边反弹。在修改代码清单 18-2 的基础上，完整的程序见代码清单 18-3。

代码清单 18-3 使用一个定时器事件让球上移和下移

```
import pygame, sys
pygame.init()
screen = pygame.display.set_mode([640,480])                    初始化
background = pygame.Surface(screen.get_size())
background.fill([255, 255, 255])

clock = pygame.time.Clock()
class Ball(pygame.sprite.Sprite):
    def __init__(self, image_file, speed, location):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location                 Ball 类定义
        self.speed = speed
    def move(self):
        if self.rect.left <= screen.get_rect().left or \
                self.rect.right >= screen.get_rect().right:
            self.speed[0] = - self.speed[0]                        ❶此行未完
        newpos = self.rect.move(self.speed)
```

```
        self.rect = newpos

my_ball = Ball('beach_ball.png', [10,0], [20, 20])
pygame.time.set_timer(pygame.USEREVENT, 1000)
direction = 1
running = True
while running:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.USEREVENT:
            my_ball.rect.centery = my_ball.rect.centery + (30*direction)
            if my_ball.rect.top <= 0 or \
            my_ball.rect.bottom >= screen.get_rect().bottom:
                direction = -direction
    clock.tick(30)
    screen.blit(background, (0, 0))
    my_ball.move()
    screen.blit(my_ball.image, my_ball.rect)
    pygame.display.flip()
pygame.quit()
```

创建 Ball 的一个实例

创建一个定时器
1000 ms = 1 秒

定时器的事件处理器

❶此行未完

完全重绘

记住，\ 是行联接符 ❶。可以用它把正常情况下应该写在一行上的内容分为两行来写。（不过不要在 \ 后面加任何空格，否则行联接符将不起作用。）

保存并运行代码清单 18-3 中的程序，应该能看到球来回移动（从一边到另一边），另外还会向上或向下移动 30 个像素（每秒移动一次）。向上或向下移动就来自定时器事件。

18.5 另一个游戏——PyPong

这一节中，我们将把前面学到的内容集中在一起（包括动画精灵、碰撞检测和事件），建立一个简单的“球拍与球”游戏，类似于 Pong。

先来看一个简单的单机版本。我们的游戏需要：

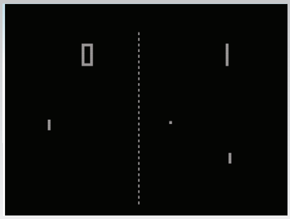
- 一个来回反弹的球；
- 一个打球的球拍；
- 一种控制球拍的方法；
- 一种记录分数并在窗口上显示分数的方法；
- 一种确定有几条“命”的方法——你有几次机会。

我们将在构建程序过程中逐个分析以上的需求。

从前的美好时光



Pong 是最早人们在家里玩的视频游戏之一。原来的 Pong 游戏没有任何软件——只是一堆电路！那时还没有家用计算机。Pong 要插入到你的电视上，你要用操纵杆来控制“球拍”。下面是这个游戏在电视屏幕上的效果图：



很少有人知道的秘密：

奶奶不仅是一个 PONG 游戏高手，还是乒乓球世界冠军呢！

球

我们之前使用的沙滩球对于 Pong 游戏来说有点大。我们需要小一点的球。卡特和我为这个游戏想出了这个有些滑稽的网球小人：



嘿，如果你被球拍打来打去，也会吓得够呛！

我们将在这个游戏中使用动画精灵，所以需要为我们的球建立一个精灵，然后为它创建一个实例。我们将使用包含 `__init__()` 和 `move()` 方法的 `Ball` 类。

```
class MyBallClass(pygame.sprite.Sprite):
    def __init__(self, image_file, speed, location):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
```

```
        self.speed = speed

    def move(self):
        self.rect = self.rect.move(self.speed)
        if self.rect.left < 0 or self.rect.right > width:
            self.speed[0] = -self.speed[0]

        if self.rect.top <= 0 :
            self.speed[1] = -self.speed[1]
```

在窗口两边反弹

在窗口顶边反弹

创建球的实例时，我们会告诉它使用哪个图像、球的速度以及球的起始位置：

```
myBall = MyBallClass('wackyball.bmp', ball_speed, [50, 50])
```

还需要把这个球增加到一个组，以便完成球和球拍之间的碰撞检测。可以创建组，同时把球增加到这个组：

```
ballGroup = pygame.sprite.Group(myBall)
```

球拍

对于球拍，我们仍然坚持 Pong 游戏的传统，只是使用一个简单的矩形。我们将要使用一个白色背景，所以把球拍创建为一个黑色矩形。也要为球拍建立一个精灵类和实例：

```
class MyPaddleClass(pygame.sprite.Sprite):
    def __init__(self, location):
        pygame.sprite.Sprite.__init__(self)
        image_surface = pygame.surface.Surface([100, 20])
        image_surface.fill([0,0,0])
        self.image = image_surface.convert()
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location

paddle = MyPaddleClass([270, 400])
```

为球拍创建一个表面

用黑色填充这个表面

将这个表面转换到一个图像

注意，对于球拍，我们并没有加载图像文件：这里只是用黑色填充一个矩形表面来创建一个图像。不过，每个精灵都需要一个 image 属性，所以我们使用 Surface.convert() 方法把表面转换为一个图像。

这个球拍只能左右移动，不能上下移动。我们让球拍的 x 位置（它的左右位置）跟着鼠标移动，所以用户可以用鼠标来控制球拍。因为这个工作在事件循环中完成，所以球拍不需要一个单独的 move() 方法。

控制球拍

上一节已经提到过，我们将用鼠标控制球拍。这里要使用 MOUSEMOTION 事件，这说明只要鼠标在 Pygame 窗口内部移动，球拍就会移动。由于鼠标在 Pygame 窗口

内时 Pygame 才能“看到”鼠标，所以球拍会自动限制在窗口的边界以内。我们将让球拍的中心跟随鼠标移动。

代码应当像这样：

```
elif event.type == pygame.MOUSEMOTION:
    paddle.rect.centerx = event.pos[0]
```

event.pos 是一个列表，包含鼠标位置的 [x, y] 值。所以 event.pos[0] 会提供鼠标移动时的 x 位置。当然，如果鼠标在左边界或右边界上，球拍会有一半在窗口之外，不过这是可以的。

还需要最后一点：球和球拍之间的碰撞检测。我们就是利用这种“碰撞”才能用球拍“打”球。出现碰撞时，只需让球的 y 速度反向（所以如果球在向下走，碰到球拍时它会反弹，开始向上移动）。代码如下：

```
if pygame.sprite.spritecollide(paddle, ballGroup, False):
    myBall.speed[1] = -myBall.speed[1]
```

还要记住每次循环时都要重绘。如果把这些内容都集中在一起，就得到了一个非常基本的类似 Pong 的程序。代码清单 18-4 给出了（至今为止）完整的代码。

代码清单 18-4　PyPong 的第一个版本

```
import pygame, sys
from pygame.locals import *

class MyBallClass(pygame.sprite.Sprite):
    def __init__(self, image_file, speed, location):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
        self.speed = speed

    def move(self):
        self.rect = self.rect.move(self.speed)
        if self.rect.left < 0 or self.rect.right > screen.get_width():
            self.speed[0] = -self.speed[0]

        if self.rect.top <= 0 :
            self.speed[1] = -self.speed[1]

class MyPaddleClass(pygame.sprite.Sprite):
    def __init__(self, location = [0,0]):
        pygame.sprite.Sprite.__init__(self)
        image_surface = pygame.surface.Surface([100, 20])
        image_surface.fill([0,0,0])
        self.image = image_surface.convert()
```

Ball 类定义

移动球（在顶边和左右两边反弹）

球拍类定义

```
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location

pygame.init()
screen = pygame.display.set_mode([640,480])
clock = pygame.time.Clock()
ball_speed = [10, 5]
myBall = MyBallClass('wackyball.bmp', ball_speed, [50, 50])
ballGroup = pygame.sprite.Group(myBall)
paddle = MyPaddleClass([270, 400])

running = True
while running:
   clock.tick(30)
   screen.fill([255, 255, 255])
   for event in pygame.event.get():
       if event.type == QUIT:
            running = False
        elif event.type == pygame.MOUSEMOTION:
            paddle.rect.centerx = event.pos[0]

    if pygame.sprite.spritecollide(paddle, ballGroup, False):
        myBall.speed[1] = -myBall.speed[1]
    myBall.move()
    screen.blit(myBall.image, myBall.rect)
    screen.blit(paddle.image, paddle.rect)
    pygame.display.flip()
pygame.quit()
```

初始化 Pygame、时钟、球和球拍

主 while 循环开始

如果鼠标移动，就移动球拍

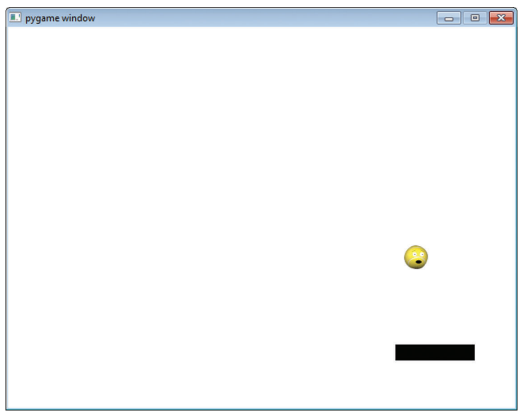
检查球是否碰到球拍

移动球

完全重绘

运行这个程序时应该能得到下面的结果。





也许吧，这可能不是最让人兴奋的游戏，不过我们只是刚刚起步，才开始在 Pygame 中编写游戏。下面再向我们的 PyPong 游戏加些东西。

记录分数并用 **pygame.font** 显示

我们要跟踪两个方面：还有几条命以及得了多少分。为了力求简单，每次球碰到窗口顶边时我们会给 1 分。另外给每个玩家 3 条命。

还需要一种方法来显示这个分数。Pygame 使用一个名为 font 的模块显示文本。可以这样来使用。

- ❒ 建立一个 font 对象，告诉 Pygame 你想要的字体样式和大小。
- ❒ 渲染文本，向字体对象传入一个字符串，它会返回一个绘制有这个文本的新的表面。
- ❒ 把这个表面块移到显示表面。

术语箱

计算机图形学中，渲染（render）是指绘制某个东西，或者让它可见。

在这里，字符串就是玩家的分数（不过首先必须把它从一个 int 转换为一个 string）。

我们需要类似下面的代码，要放在代码清单 18-4 中的事件循环前面（而且要在 paddle=MyPaddleClass([270,400]) 代码行后面）：

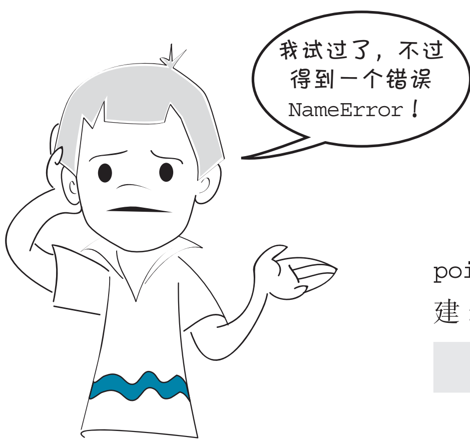
```
score_font = pygame.font.Font(None, 50)        ← 创建字体对象
score_surf = score_font.render(str(score), 1, (0, 0, 0))   ← 渲染文本到表面块 Score-surf
score_pos = [10, 10]                           ← 设置文本位置
```

第一行中的第一个参数（这里是 None）可以告诉 Pygame 我们希望使用什么字体（类型样式）。通过传入 None，就是在告诉 Pygame 要使用一个默认字体。

然后，在事件循环内部，我们需要这样的代码：

```
screen.blit(score_surf, score_pos)   ← 把含有分数文本的表面块移到这个位置
```

这样每次循环时都会重绘分数文本。



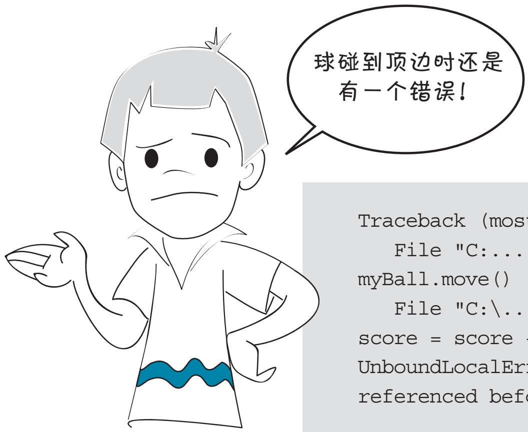
当然了，卡特，我们还没有创建 points 变量（我正打算这么做呢）。在创建 font 对象的代码前面增加这样一行代码：

```
score = 0
```

现在，要跟踪分数……因为我们已经用球的 move() 方法检测了球什么时候碰到窗口的顶边（来完成反弹），所以只需要在这里再增加几行：

```
if self.rect.top <= 0 :
    self.speed[1] = -self.speed[1]
    score = score + 1
    score_surf = score_font.render(str(score), 1, (0, 0, 0))
```

两行新代码




```
Traceback (most recent call last):
   File "C:...", line 59, in <module>
myBall.move()
   File "C:\...", line 24, in move
score = score + 1
UnboundLocalError: local variable 'score'
referenced before assignment
```

唉呀！我们忘记命名空间的问题了。还记得第 15 章中那个又大又长的解释吗？现在可以看到命名空间的一个实际例子了。尽管我们确实有一个名为 score 的变量，但是这里试图从 Ball 类的 move() 方法中使用这个变量。这个类在寻找一个名为 score 的局部变量，而这个局部变量并不存在。实际上，我们希望使用先前已经创建的全局变量，所以只需要告诉 move() 方法使用全局变量 score，如下：

```
def move(self):
    global score
```

还要让 score_font（分数的 font 对象）和 score_surf（包含渲染文本的表面块）作为全局变量，因为它们是用 move() 方法更新的。所以代码实际上应当像这样：

```
def move(self):
    global score, score_font, score_surf
```

现在应该能正常工作了！再试试看。应该能看到窗口左上角的分数，而且当你把球弹到窗口顶边时这个分数应该会增加。

跟踪还有几条命

现在来跟踪还有几条命。对目前来说，如果漏了球，它就会从窗口底边掉下去，再也看不到了。我们希望给玩家 3 条命或者 3 个机会，所以下面建立一个名为

lives 的变量，把它设置为 3。

```
lives = 3
```

玩家漏了球而且球掉到窗口底边后，要将 lives 减 1，等待几秒，然后重新开始，又提供一个新球：

```
if myBall.rect.top >= screen.get_rect().bottom:
    lives = lives - 1
    pygame.time.delay(2000)
    myBall.rect.topleft = [50, 50]
```

这个代码要放在 while 循环中。顺便说一句，为什么对于球我们会写成 myBall.rect，而对于 screen 要写为 get_rect() 呢？这有下面几个原因。

- myBall 是一个动画精灵，动画精灵都包含一个 rect。
- screen 是一个表面，而表面不包含 rect。可以用 get_rect() 函数找到包围一个表面的 rect。

如果做了上述修改，并运行程序，你会看到玩家现在有 3 条命。

增加一个生命计数器

很多游戏会给玩家多条命，大多数这样的游戏都会采用某种方法显示还剩下几条命。我们这个游戏也可以做到这一点。

一种简单的方法是显示一些球，剩几条命就显示几个球。可以把这些球放在右上角。以下是画出生命计数器的 for 循环中使用的小公式：

```
for i in range (lives):
    width = screen.get_rect().width
    screen.blit(myBall.image, [width - 40 * i, 20])
```

这个代码也要放在主 while 循环中，应当放在事件循环前面（但要在 screen.blit(score_text, textpos) 代码行之后）。

游戏结束

最后还需要增加一点：当玩家丢掉最后一条命时要显示一个“游戏结束”的消息。我们要建立两个字体对象，分别包含我们的消息和玩家的最后分数，渲染这两个文本（创建绘有文本的表面），再将这些表面块移到 screen。

另外还要在最后一局结束后避免球再次出现。为了做到这一点，要建立一个 done 变量告诉我们何时游戏结束。运行在主 while 循环中的以下代码会完成这项工作。

```
if myBall.rect.top >= screen.get_rect().bottom:
    lives = lives - 1
    if lives == 0:
        final_text1 = "Game Over"
        final_text2 = "Your final score is:  " + str(score)
        ft1_font = pygame.font.Font(None, 70)
        ft1_surf = ft1_font.render(final_text1, 1, (0, 0, 0))
        ft2_font = pygame.font.Font(None, 50)
        ft2_surf = ft2_font.render(final_text2, 1, (0, 0, 0))
        screen.blit(ft1_surf, [screen.get_width()/2 - \
                    ft1_surf.get_width()/2, 100])
        screen.blit(ft2_surf, [screen.get_width()/2 - \
                    ft2_surf.get_width()/2, 200])
        pygame.display.flip()
        done = True
    else:  #wait 2 seconds, then start the next ball
        pygame.time.delay(2000)
        myBall.rect.topleft = [(screen.get_rect().width) - 40*lives, 20]
```

如果球碰到底边就减一条命

行联接符

将文本在窗口中居中放置

把所有这些内容集中在一起，可以得到最终的 PyPong 程序，如代码清单 18-5 所示。

代码清单 18-5　最终的 PyPong 代码

```
import pygame, sys

class MyBallClass(pygame.sprite.Sprite):
    def __init__(self, image_file, speed, location):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
        self.speed = speed

    def move(self):
        global score, score_surf, score_font
        self.rect = self.rect.move(self.speed)
        if self.rect.left < 0 or self.rect.right > screen.get_width():
            self.speed[0] = -self.speed[0]

        if self.rect.top <= 0 :
            self.speed[1] = -self.speed[1]
            score = score + 1
            score_surf = score_font.render(str(score), 1, (0, 0, 0))

class MyPaddleClass(pygame.sprite.Sprite):
    def __init__(self, location = [0,0]):
        pygame.sprite.Sprite.__init__(self)
        image_surface = pygame.surface.Surface([100, 20])
        image_surface.fill([0,0,0])
```

定义 Ball 类

定义球拍类

```
        self.image = image_surface.convert()
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location

pygame.init()
screen = pygame.display.set_mode([640,480])
clock = pygame.time.Clock()
myBall = MyBallClass('wackyball.bmp', [10,5], [50, 50])
ballGroup = pygame.sprite.Group(myBall)
paddle = MyPaddleClass([270, 400])
lives = 3
score = 0
score_font = pygame.font.Font(None, 50)
score_surf = score_font.render(str(score), 1, (0, 0, 0))
score_pos = [10, 10]
done = False
running = True
while running:
    clock.tick(30)
    screen.fill([255, 255, 255])
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEMOTION:
            paddle.rect.centerx = event.pos[0]
    if pygame.sprite.spritecollide(paddle, ballGroup, False):
        myBall.speed[1] = -myBall.speed[1]
    myBall.move()
    if not done:
        screen.blit(myBall.image, myBall.rect)
        screen.blit(paddle.image, paddle.rect)
        screen.blit(score_surf, score_pos)
        for i in range (lives):
            width = screen.get_width()
            screen.blit(myBall.image, [width - 40 * i, 20])
        pygame.display.flip()
    if myBall.rect.top >= screen.get_rect().bottom:
        lives = lives - 1
        if lives == 0:
            final_text1 = "Game Over"
            final_text2 = "Your final score is:  " + str(score)
            ft1_font = pygame.font.Font(None, 70)
            ft1_surf = ft1_font.render(final_text1, 1, (0, 0, 0))
            ft2_font = pygame.font.Font(None, 50)
            ft2_surf = ft2_font.render(final_text2, 1, (0, 0, 0))
            screen.blit(ft1_surf, [screen.get_width()/2 - \
                        ft1_surf.get_width()/2, 100])
            screen.blit(ft2_surf, [screen.get_width()/2 - \
                        ft2_surf.get_width()/2, 200])
            pygame.display.flip()
            done = True
        else:
            pygame.time.delay(2000)
            myBall.rect.topleft = [50, 50]
pygame.quit()
```

定义球拍类

初始化

创建 font 对象

主程序（while 循环）开始

检测鼠标运动，移动球拍

检测球与球拍之间的碰撞

移动球

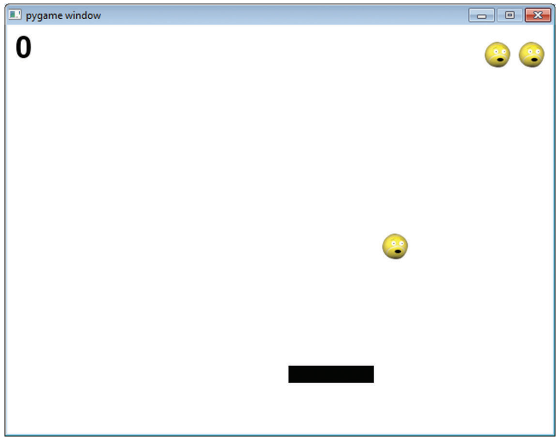
完全重绘

如果球碰到底边就减一条命

创建和绘制最终的分数文本

2 秒之后，获得新的一条命，重新开始

如果运行代码清单 18-5 中的代码，应该能看到这样的结果。



如果在编辑器中注意观察，可以看到这大约有 75 行代码（加上一些空行）。这是目前为止我们创建的最大的程序了，虽然运行时看起来很简单，但却包含了丰富的内容。

下一章，我们将要学习 Pygame 中的声音，另外还会向这个 PyPong 游戏添加一些声音。

你学到了什么

在这一章，你学到了以下内容。

- 事件。
- Pygame 事件循环。
- 事件处理。
- 键盘事件。
- 鼠标事件。
- 定时器事件（以及用户事件类型）。
- `pygame.font`（用于向 Pygame 程序添加文本）。
- 把所有内容集中在一起建立一个游戏！

测试题

1. 程序可以响应哪两种事件？
2. 处理事件的代码叫什么？

3. Pygame 检测按键时使用的事件类型名是什么？
4. `MOUSEMOVE` 事件的哪个属性指出了鼠标位于窗口的哪个位置？
5. 如何找出 Pygame 中下一个可用的事件编号（例如，如果你想添加一个用户事件）？
6. 如何创建一个定时器在 Pygame 中生成定时器事件？
7. 在 Pygame 窗口中显示文本时要使用什么对象？
8. 要让文本出现在一个 Pygame 窗口中，需要哪 3 个步骤？

动手试一试

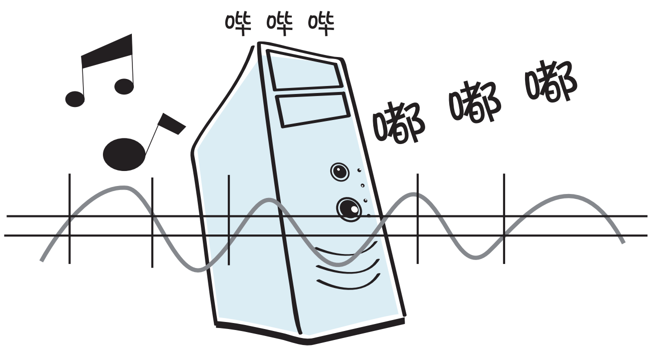
1. 如果球没有碰到球拍的顶边，而是碰到了球拍的左右两边，有没有什么奇怪的现象发生？它会在球拍中间持续反弹一段时间。你明白这是为什么吗？你能解决这个问题吗？我在后面的答案中给出了一个解决方案，不过在看答案之前你自己先试试看。
2. 试着重写这个程序（代码清单 18-4 或代码清单 18-5），让球的反弹有点随机性。可以改变球在球拍或墙上反弹的方式，使用随机的速度，或者也可以采用你能想到的其他做法。（我们在第 15 章见过 `random.randint()` 和 `random.random()`，所以你应该知道如何生成随机数，包括整数和浮点数。）

第 19 章

声　　音

上一章中，我们使用之前学到的关于图形、动画精灵、碰撞、动画和事件的知识建立了我们的第一个图形游戏 PyPong。这一章将会再增加一个内容：声音。为了让程序更有趣、更好玩，视频游戏和很多其他程序都使用了声音。

声音既可以作为输入，也可以作为输出。作为输入，需要把一个麦克风或其他音源连接到计算机，程序会把声音记录下来，或者对它做其他处理（可能通过互联网发送）。不过声音作为输出更为常见，这也是这本书要讨论的内容。我们将学习如何播放音乐或音效等声音，以及如何把它们添加到程序中（比如 PyPong）。



19.1　从 Pygame 寻求更多帮助——mixer

有些内容可能很复杂，比如图形，声音也是如此，因为不同的计算机播放声音的硬件和软件不同。为了让问题简单一些，我们还是打算从 Pygame 寻求一些帮助。

Pygame 有一个处理声音的模块，名为 `pygame.mixer`。在真实世界中，取不同的声音并把它们混合在一起的设备叫做“混音器”（mixer），Pygame 中的模块也正是因此得名。

19.2 制造声音与播放声音

程序产生声音有两种基本方式。程序可以生成或合成声音——这是指制造不同音高和音量的声波来从头创建声音。或者程序也可以播放一段录制的声音。这可以是 CD 上的一段音乐、一个 MP3 声音文件，或者其他类型的声音文件。

在这本书中，我们只学习如何播放声音。要从零开始制作我们自己的声音，这个主题涵盖的内容太多，而这本书的篇幅有限，根本没办法详细介绍。如果你对计算机生成的声音感兴趣，目前有很多程序可以利用，这些程序能够从计算机生成音乐和声音。

19.3 播放声音

播放声音时，要从硬盘（或从 CD，或者有时从互联网）得到一个声音文件，把它转换成可以在计算机的扬声器或耳机上听到的声音。计算机上可以使用多种不同类型的声音文件。以下是比较常见的类型。

- 波形文件——文件名以 .wav 结尾，如 hello.wav。
- MP3 文件——文件名以 .mp3 结尾，如 mySong.mp3。
- WMA（Windows 媒体音频，Windows Media Audio）文件——文件名以 .wma 结尾，如 someSong.wma。
- Ogg Vorbis 文件——文件名以 .ogg 结尾，如 yourSong.ogg。

我们的例子中打算使用 .wav 和 .mp3 文件。我们将要使用的所有声音都放在这本书安装目录下的 \sounds 文件夹中。例如，在 Windows 计算机上，就应该在 c:\Program Files\HelloWorld\examples\sounds 里。

在程序中包含一个声音文件有两种方法。可以把声音文件复制到保存程序的同一个文件夹中。Python 会在这里查找文件，所以可以在程序中直接使用这个文件的名，例如：

```
sound_file = "my_sound.wav"
```

如果声音文件没有复制到程序所在的同一个文件夹中，就必须把声音文件的位置明确地告诉 Python，例如：

```
sound_file = "c:\Program Files\HelloWorld\sounds\my_sound.wav"
```

举几个例子，假设你已经把声音文件复制到保存程序的文件夹。这说明，只要在例子中用到声音文件，你只会看到文件名，而不是文件的完整位置。如果声音文

件没有复制到程序文件夹，就要把文件名替换为完整的文件位置。

> 如果使用这本书的安装程序来安装，那么这些例子的所有声音文件都已经在你的硬盘上了。否则，可以在本书的网站（www.helloworldbook2.com.）上找到这些声音文件。

启动 pygame.mixer

要播放声音，首先必须初始化（initialize）`pygame.mixer`。还记得初始化是什么意思吗？指的是开始时让某个东西做好准备。

让 `pygame.mixer` 做好准备很容易。只需要在初始化 Pygame 之后增加一行代码：

```
pygame.mixer.init()
```

所以，使用 Pygame 处理声音的程序中最前面几行代码应该像这样：

```
import pygame
pygame.init()
pygame.mixer.init()
```

现在我们已经做好准备可以播放声音了。这些程序主要使用两种类型的声音。第一种是音效或声音片段。这些声音往往很短，通常保存在 .wav 文件中。对于这种类型的声音，`pygame.mixer` 会使用一个 `Sound` 对象，如下：

```
splat = pygame.mixer.Sound("splat.wav")
splat.play()
```

另一种大量使用的声音是音乐。音乐大多存储在 .mp3、.wma 或 .ogg 文件中。要播放这些音乐，Pygame 会使用 `mixer` 中的 `music` 模块。可以这样来使用：

```
pygame.mixer.music.load("bg_music.mp3")
pygame.mixer.music.play()
```

这样歌曲（或音乐文件里的任何音乐）会播放一次，然后停止。



下面来试着播放一些声音。首先来播放“啪啪”声。

我们还需要一个 while 循环来保证 Pygame 程序一直运行。另外，尽管目前没有画任何图形，但 Pygame 程序仍然需要有一个窗口。而且，在某些系统上，mixer 初始化还需要一点时间。如果播放声音太快，你可能只能听到声音的一部分，或者根本什么都听不到。所以我们会等一会，直到 mixer 准备好。这个代码见代码清单 19-1。

代码清单 19-1　尝试在 Pygame 中播放声音

```
import pygame, sys
pygame.init()
pygame.mixer.init()

screen = pygame.display.set_mode([640,480])
pygame.time.delay(1000)

splat = pygame.mixer.Sound("splat.wav")
splat.play()

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

初始化 Pygame 和 mixer
创建一个 Pygame 窗口
等 1 秒钟让 mixer 完成初始化
创建声音对象
播放声音
Pygame 事件循环

试着运行这个程序，看看它的效果如何。

现在来使用 mixer.music 模块播放一些音乐。只需要修改代码清单 19-1 中的几行代码。新代码见代码清单 19-2。

代码清单 19-2　播放音乐

```
import pygame, sys
pygame.init()
pygame.mixer.init()

screen = pygame.display.set_mode([640,480])
pygame.time.delay(1000)

pygame.mixer.music.load("bg_music.mp3")
pygame.mixer.music.play()

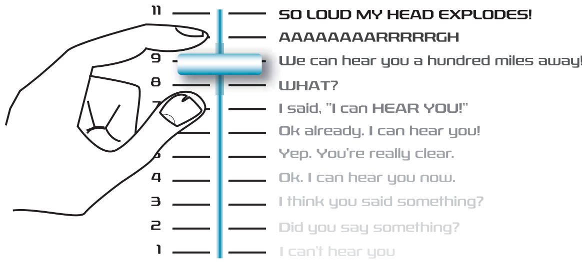
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

修改这两行代码

再来试一试，确保你能听到音乐。
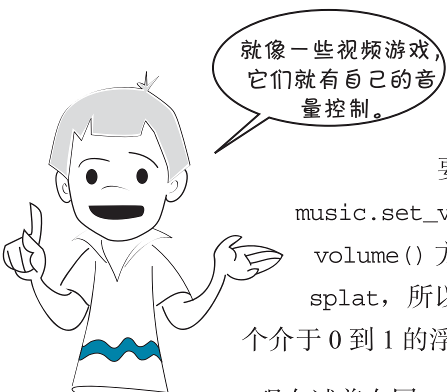
我不知道你的具体情况，不过对我来说听起来声音太大了。我必须把计算机的音量调小。下面来看如何在程序中控制声音的音量。

19.4 控制音量



可以使用音量控制开关来控制计算机上的声音音量。在 Windows 系统上，这是利用系统托盘里的小扬声器图标完成的。这个设置会控制计算机上所有声音的音量。你的扬声器本身可能也有一个音量控制杆。

不过除此以外，我们还可以控制 Pygame 发送到计算机声卡的音量。



好在我们可以单独控制每个声音的音量，例如，可以让音乐音量小一些，让“啪啪”声更响一些。

要设置音乐的音量，需要使用 pygame.mixer.music.set_volume()。而每个声音对象都有一个 set_volume() 方法。在第一个例子中，声音对象的名字是 splat，所以我们使用了 splat.set_volume()。音量是一个介于 0 到 1 的浮点数；例如，0.5 就是最大音量的 50% 或一半。

现在试着在同一个程序中播放音乐和声音。先来播放一首歌曲，在最后再播放“啪啪”声。还要把声音的音量调低一下。我们把音乐的音量设置为 30%，“啪啪”声的音量为 50%。这个代码见代码清单 19-3。

代码清单 19-3 有音量调节的音乐和声音

```
import pygame, sys
pygame.init()
pygame.mixer.init()
screen = pygame.display.set_mode([640,480])
pygame.time.delay(1000)
pygame.mixer.music.load("bg_music.mp3")
pygame.mixer.music.set_volume(0.30)          ← 调节音乐的音量
pygame.mixer.music.play()
splat = pygame.mixer.Sound("splat.wav")
splat.set_volume(0.50)          ← 调节音效的音量
splat.play()
running = True
```

```
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
pygame.quit()
```

试着运行这个程序，看看它的效果。



卡特注意到这样一个问题：程序一旦开始播放音乐，就会继续做下一件事，在这里就是播放“啪啪”声。为什么会出现这种情况呢？原因是：通常我们都是使用背景音乐，你肯定不希望程序只是“呆坐在那里”，一直等到整首歌都播放完之后才开始做事情。在下一节中，我们会让它按我们希望的方式工作。

播放背景音乐

背景音乐是指玩游戏时在背景播放的音乐。所以一旦开始播放背景歌曲，Pygame 必须做好准备来做其他事情，比如移动动画精灵，或者检查是否有鼠标和键盘输入。它不会一直等到歌曲播放完。

但是如果你想知道歌曲什么时候结束该怎么做呢？你可能希望等这首歌播放完就播放另一首歌或者另一个声音（就像我们现在要做的一样）。你怎么知道音乐什么时候结束呢？为此，Pygame 提供了一种方法：你可以询问 `mixer.music` 模块是否还在忙于播放一首歌。如果忙，就能知道歌曲还没有播放完。如果它不忙，说明歌曲已经结束。下面就来试一试。

要查看 `music` 模块是否在忙于播放一首歌，可以使用 `mixer.music` 模块的 `get_busy()` 函数。如果它仍在忙，这个函数会返回值 `True`；如果不忙，函数会返回 `False`。这一次，我们要让程序先播放歌曲，然后播放音效，再自动结束程序。代码清单 19-4 显示了如何完成这些工作。

代码清单 19-4 等待歌曲结束

```
import pygame, sys
pygame.init()
pygame.mixer.init()
```

```
screen = pygame.display.set_mode([640,480])
pygame.time.delay(1000)

pygame.mixer.music.load("bg_music.mp3")
pygame.mixer.music.set_volume(0.3)
pygame.mixer.music.play()
splat = pygame.mixer.Sound("splat.wav")
splat.set_volume(0.5)
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    if not pygame.mixer.music.get_busy():      ← 检查音乐是否播放完毕
        splat.play()
        pygame.time.delay(1000)                ← 等待 1 秒钟让“啪啪”声结束
        running = False
pygame.quit()
```

这个代码会播放一次歌曲，接下来播放音效，然后程序会结束。

19.5　重复音乐

如果要使用一首歌作为游戏的背景音乐，你可能希望只要程序在运行，音乐就一直继续下去。music 模块可以做到这一点。可以让音乐重复播放一定的次数，比如：

```
pygame.mixer.music.play(3)
```

这会让歌曲播放 3 次。

> Pygame 文档中称，pygame.mixer.music.play(3) 会播放 4 次歌曲：第一次再加 3 次重复。这一点他们弄错了。这行代码只会播放 3 次歌曲。

还可以传入一个特殊值 –1，这会让歌曲永远重复下去，如下：

```
pygame.mixer.music.play(-1)
```

这样一来，歌曲会一直重复播放，或者只要 Pygame 程序在运行就一直播放。（实际上，不一定非得是 –1。任何负数都可以达到这个目的。）

19.6　为 PyPong 增加声音

我们已经了解了播放声音的基础知识，下面向我们的 PyPong 游戏添加一些声音。

首先，每次球碰到球拍时要增加一个声音。我们已经知道球什么时候碰到球拍，

因为前面使用了碰撞检测，当球碰到球拍时要让它反向。应该记得代码清单 18-5 中的代码：

```
if pygame.sprite.spritecollide(paddle, ballGroup, False):
    myBall.speed[1] = -myBall.speed[1]
```

现在需要增加代码播放声音。我们需要在程序最前面增加一行 pygame.mixer.init()，还要创建声音对象以备使用：

```
hit = pygame.mixer.Sound("hit_paddle.wav")
```

另外还要设置音量，让声音不至于太吵：

```
hit.set_volume(0.4)
```

当球碰到球拍时，播放这个声音：

```
if pygame.sprite.spritecollide(paddle, ballGroup, False):
    myBall.speed[1] = -myBall.speed[1]
    hit.play()        ← 播放声音
```

把这个代码添加到代码清单 18-5 的 PyPong 程序中。一定要把 hit_paddle.wav 文件复制到保存程序的同一个位置。运行这个程序时，每次球碰到球拍时你都会听到一个声音。



19.7　更多声音

球碰到球拍时会发出 hit 声音，下面再增加另外一些声音。我们将为下面这些情况添加声音：

- 球碰到两边的墙时；
- 球碰到顶边而且玩家得分时；
- 玩家漏球，球碰到底边时；
- 新的一条命开始时；
- 游戏结束时。

首先需要为所有这些情况创建声音对象。可以把相应代码放在 pygame.mixer.init() 之后但在 while 循环之前的任何位置上。

```
hit_wall = pygame.mixer.Sound("hit_wall.wav")
hit_wall.set_volume(0.4)
get_point = pygame.mixer.Sound("get_point.wav")
get_point.set_volume(0.2)
splat = pygame.mixer.Sound("splat.wav")
splat.set_volume(0.6)
new_life = pygame.mixer.Sound("new_life.wav")
new_life.set_volume(0.5)
bye = pygame.mixer.Sound("game_over.wav")
bye.set_volume(0.6)
```

这里我选择了不同的音量，这只是为了看看哪种音量听起来最合适。可以按你的喜好来设置音量。另外要记住，所有声音文件都要复制到保存代码的位置上。这些声音都可以在 \examples\sounds 文件夹或者本书网站上找到。

现在需要在发生这些事件时的相应代码中增加 play() 方法。只要碰到窗口左右两边就应当发出 hit_wall 声音。这个事件在球的 move() 法中检测，我们还要让球的 x 速度反向（使球在两边"反弹"）。在原来的代码清单 18-5 中，这是第 14 行（if self.rect.left < 0 or self.rect.right > screen.get_width():）。

所以，在反向时还可以播放声音。代码如下：

```
if self.rect.left < 0 or self.rect.right > screen.get_width():
    self.speed[0] = -self.speed[0]
    hit_wall.play()          ← 碰到两边的墙时播放声音
```

可以对 get_point 声音做同样的处理。在球的 move() 方法下面一点点的地方，我们检测了球是否碰到窗口顶边。在这里要让球反弹，并为玩家加 1 分。现在还要播放一个声音。新代码如下：

```
if self.rect.top <= 0 :
    self.speed[1] = -self.speed[1]
    points = points + 1
    score_text = font.render(str(points), 1, (0, 0, 0))
    get_point.play()     ← 得分时播放声音
```

增加这些代码后，试着运行程序，看看有什么效果。

接下来继续增加代码，当玩家漏球而丢掉一条命时会播放一个声音。这个事件在主 while 循环中检测，也就是原来的代码清单 18-5 中的第 63 行（if myBall.rect.top >= screen.get_rect().bottom:）。只需再增加这样一行代码：

```
if myBall.rect.top >= screen.get_rect().bottom:
        splat.play()
        # lose a life if the ball hits the bottom
        lives = lives - 1
```

当漏球而丢掉一条命时播放声音

还可以在新的一条命开始时增加一个声音。这种情况在代码清单 18-5 的最后 3 行代码发生（也就是 else 块中）。这一次我们会在开始新的一条命之前留出一点时间来播放音效：

```
else:
   pygame.time.delay(1000)
   new_life.play()
   myBall.rect.topleft = [50, 50]
   screen.blit(myBall.image, myBall.rect)
   pygame.display.flip()
   pygame.time.delay(1000)
```

与原来的程序不同，现在不是等待 2 秒，我们会等待 1 秒（1000 毫秒），再播放声音，然后在开始下一轮之前再等待 1 秒。可以试试看，听听效果如何。

这里还要增加一个音效：游戏结束时需要播放一个声音。这种情况在代码清单 18-5 的第 65 行发生（if lives == 0:）。在这里增加下面这行代码，就会播放 bye 声音：

```
if lives == 0:
    bye.play()
```

试试看效果如何。



唉呀！我们忘了一件事。播放 bye 声音和 splat 声音的代码放在主 while 循环中，Pygame 窗口关闭前它们是不会停止的，所以只要 while 循环在运行，声音就会反复播放！需要增加一些代码来确保它只会播放一次。

这里就要使用前面定义的 done 变量了，它会告诉我们游戏何时结束。可以把代码修改成下面这样：

```
if myBall.rect.top >= screen.get_rect().bottom:
        if not done:
            splat.play()
        lives = lives - 1
        if lives == 0:
            if not done:
                bye.play()
```

确保声音只播放一次

试试看，确认程序能正常工作。



嗯……这个问题可能需要稍稍考虑一下。这里通过 done 变量来告诉我们游戏何时结束，利用这一点，我们能够知道什么时候播放 bye 声音，以及什么时候显示最后的分数消息。不过这球在干嘛？

尽管球已经到达窗口底边，但它仍然在不停地移动！球在继续向下走，越走越远，没有什么来阻止它，所以它的 y 值会越来越大。虽然它在屏幕底边的“下面”，我们看不到，但是仍然能够听到它的声音！球仍在移动，所以当球的 x 值变得足够大或者足够小时，它还会在“左右两边”上反弹。这种情况发生在 move() 方法中，只要 while 循环在运行，这个方法就会一直运行。

怎么解决这个问题呢？我们可以采用以下几种办法。

- 游戏结束时把球的速度设置为 [0,0] 来阻止球继续移动。
- 查看球是否在窗口底边以下，如果是，就不再播放 hit_wall 声音。
- 检查 done 变量，如果游戏已经结束就不再播放 hit_wall 声音。

我选择了第二种方法，不过上面这几种方法都能奏效。你可以选择以上的任意一种方法，修改你的代码来解决这个问题。

19.8 为 PyPong 添加音乐

还有一件事要做，就是添加音乐。需要加载音乐文件，设置音量，然后开始播放。我们希望玩游戏期间音乐一直在重复，所以会使用特殊值 –1，如下：

```
pygame.mixer.music.load("bg_music.mp3")
pygame.mixer.music.set_volume(0.3)
pygame.mixer.music.play(-1)
```

这个代码可以放在主 while 循环前面的任意位置。它会开始播放音乐。现在只需要在最后让音乐停下来，有一个很好的办法来做到这一点。pygame.mixer.music 有一个 fadeout() 方法，会让音乐淡出（逐渐减弱直到消失），而不是戛然而止。只需要告诉它淡出需要多长时间，例如：

```
pygame.mixer.music.fadeout(2000)
```

这里设置为 2000 毫秒，也就是 2 秒。这一行可以与 done = True 设置放在同一个位置。（这个设置在前在后都无关紧要。）

现在程序已经增加了音效和音乐。试试看听起来怎么样！也许你想看看如何把所有这些内容整合在一起，下面给出这个程序的最后版本，也就是代码清单 19-5。一定要确保 wackyball.bmp 和所有声音文件与程序在同一个文件夹中。

代码清单 19-5 有声音和音乐的 PyPong

```
import pygame, sys

class MyBallClass(pygame.sprite.Sprite):
    def __init__(self, image_file, speed, location = [0,0]):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
        self.speed = speed

    def move(self):
        global points, score_text
        self.rect = self.rect.move(self.speed)
        if self.rect.left < 0 or self.rect.right > screen.get_width():
            self.speed[0] = -self.speed[0]
            if self.rect.top < screen.get_height():
                hit_wall.play()                    ← 球碰到两边的墙时播放声音

        if self.rect.top <= 0 :
            self.speed[1] = -self.speed[1]
            points = points + 1
            score_text = font.render(str(points), 1, (0, 0, 0))
            get_point.play()                ← 球碰到顶边（玩家得分）时播放声音
```

```
class MyPaddleClass(pygame.sprite.Sprite):
    def __init__(self, location = [0,0]):
        pygame.sprite.Sprite.__init__(self)
        image_surface = pygame.surface.Surface([100, 20])
        image_surface.fill([0,0,0])
        self.image    = image_surface.convert()
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.top = location
pygame.init()                                  初始化 Pygame 的 sound 模块
pygame.mixer.init()
                                               加载音乐文件

pygame.mixer.music.load("bg_music.mp3")
pygame.mixer.music.set_volume(0.3)             设置音乐的音量
pygame.mixer.music.play(-1)
hit = pygame.mixer.Sound("hit_paddle.wav")
hit.set_volume(0.4)                            开始播放音乐，一直重复
new_life = pygame.mixer.Sound("new_life.wav")
new_life.set_volume(0.5)
splat = pygame.mixer.Sound("splat.wav")        创建声音对象，加载声音，
splat.set_volume(0.6)                          并设置每个声音的音量
hit_wall = pygame.mixer.Sound("hit_wall.wav")
hit_wall.set_volume(0.4)

get_point = pygame.mixer.Sound("get_point.wav")
get_point.set_volume(0.2)
bye = pygame.mixer.Sound("game_over.wav")
bye.set_volume(0.6)
screen = pygame.display.set_mode([640,480])
clock = pygame.time.Clock()

myBall = MyBallClass('wackyball.bmp', [12,6], [50, 50])
ballGroup = pygame.sprite.Group(myBall)
paddle = MyPaddleClass([270, 400])
lives = 3
points = 0

font = pygame.font.Font(None, 50)
score_text = font.render(str(points), 1, (0, 0, 0))
textpos = [10, 10]
done = False

running = True
while running:
    clock.tick(30)
    screen.fill([255, 255, 255])
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEMOTION:
            paddle.rect.centerx = event.pos[0]         球碰到球拍时播放声音

    if pygame.sprite.spritecollide(paddle, ballGroup, False):
        hit.play()
        myBall.speed[1] = -myBall.speed[1]
```

```
    myBall.move()

    if not done:
        screen.blit(myBall.image, myBall.rect)
        screen.blit(paddle.image, paddle.rect)
        screen.blit(score_text, textpos)
        for i in range (lives):
            width = screen.get_width()
            screen.blit(myBall.image, [width - 40 * i, 20])
        pygame.display.flip()

    if myBall.rect.top >= screen.get_rect().bottom:
        if not done:
            splat.play()                  ◁—— 玩家丢一条命时播放声音
        lives = lives - 1
        if lives <= 0:
            if not done:
                pygame.time.delay(1000)   | 等待 1 秒，然后播放结束声音
                bye.play()                |
            final_text1 = "Game Over"
            final_text2 = "Your final score is:  " + str(points)
            ft1_font = pygame.font.Font(None, 70)
            ft1_surf = font.render(final_text1, 1, (0, 0, 0))
            ft2_font = pygame.font.Font(None, 50)

            ft2_surf = font.render(final_text2, 1, (0, 0, 0))
            screen.blit(ft1_surf, [screen.get_width()/2 - \
                        ft1_surf.get_width()/2, 100])
            screen.blit(ft2_surf, [screen.get_width()/2 - \
                        ft2_surf.get_width()/2, 200])

            pygame.display.flip()
            done = True
            pygame.mixer.music.fadeout(2000)      ◁—— 音乐淡出
        else:
            pygame.time.delay(1000)       | 开始新的一条命时播放声音
            new_life.play()               |
            myBall.rect.topleft = [50, 50]
            screen.blit(myBall.image, myBall.rect)
            pygame.display.flip()
            pygame.time.delay(1000)
pygame.quit()
```

这个代码太长了！（大约 100 行，还要加上一些空行。）这个程序完全可以写得短一些，不过那样一来，读代码和理解起来都会更困难。其实这几章我们一直都在构建这个程序，每章补充一点内容，所以你并不需要一次键入所有这些代码。

如果你是按顺序读这本书，现在应该已经了解程序的各个部分分别做什么，也应该知道这些部分如何整合到一起。不过万一你需要这个程序的完整代码，也可以在 \examples 文件夹（如果已经安装本书的安装程序的话）和网站上找到这个程序的代码清单。

在下一章，我们将建立一个不同类型的图形程序：一个有按钮、菜单的程序，也就是是一个 GUI。

你学到了什么

在这一章，你学到了以下内容。

- 如何向程序添加声音。
- 如何播放声音片段（通常是 .wav 文件）。
- 如何播放音乐文件（通常是 .mp3 文件）。
- 如何知道一个声音已经播放完毕。
- 如何控制音效和音乐的音量。
- 如何让音乐重复，使它反复播放。
- 如何让音乐淡出。

测试题

1. 可以用哪几种类型的文件存储声音？
2. 哪个 Pygame 模块用来播放音乐？
3. 如何设置一个 Pygame 声音对象的音量？
4. 如何设置背景音乐的音量？
5. 如何让音乐淡出？

动手试一试

试着向第 1 章中的猜数游戏添加声音。尽管这个游戏是文本模式的，但与这一章中的例子一样，仍然需要增加一个 Pygame 窗口。\examples\sounds 文件夹（和网站上）有一些声音可供你使用：

- Ahoy.wav
- TooLow.wav
- TooHigh.wav
- WhatsYerGuess.wav
- AvastGotIt.wav
- NoMore.wav

或者，你也可以录制自己的声音，这可能很有意思。可以使用一个录音工具，比如 Windows 中的 Sound Recorder，或者可以从 audacity.sourceforge.net/ 下载一个免费程序 Audacity（很多操作系统上都提供了这个工具）。

第 20 章
更 多 GUI

第 6 章我们已经建立了一些简单的 GUI，那时我们使用 EasyGui 来建立一些对话框。不过 GUI 需要的不只是对话框。大多数现代程序中，整个程序都在一个 GUI 中运行。这一章中，我们将了解如何使用 PyQt 建立 GUI，它能为你提供更多灵活性，可以对程序的外观有更多控制。

PyQt 模块可以帮助我们创建 GUI。我们首先使用这个模块来建立一个新版本的温度转换程序。

20.1　使用 PyQt

使用 PyQt 之前，必须确保你的计算机上已经安装有这个模块。如果你使用这本书的安装程序来安装 Python，就已经安装有 PyQt。不然，还得另外下载安装。PyQt 可以从 www.riverbankcomputing.com/software/Pyqt/download 得到。要根据你的操作系统和所使用的 Python 版本得到正确的 PyQt 版本（如果运行这本书的安装程序，PyQt 的版本是 2.7.3）。我们这里使用 PyQt 4.1。

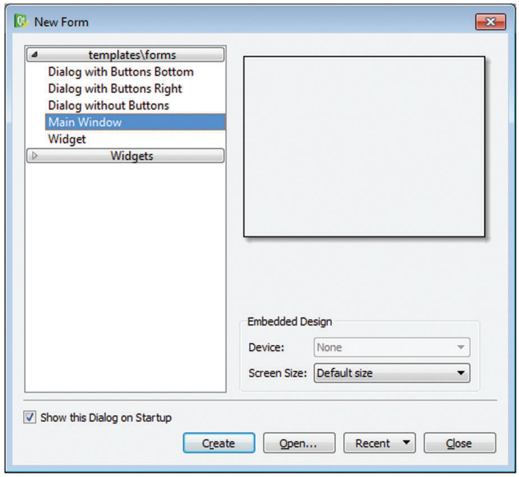
PyQt 还有一个版本是 PyQt 5，它和 PyQt 4 很不一样。如果你没有使用我们的安装程序，请确保你安装的是 PyQt 4。

写一个 GUI 程序大体上可以分为两个主要部分。你需要创建用户界面本身（UI），然后编写代码让 UI 按照你的想法来实现功能。创建 UI 需要在窗口上排布一些东西，比如按钮、文本框、选择框等。然后你编写代码来响应按钮点击、文本框输入、选择框中选择某项等动作。

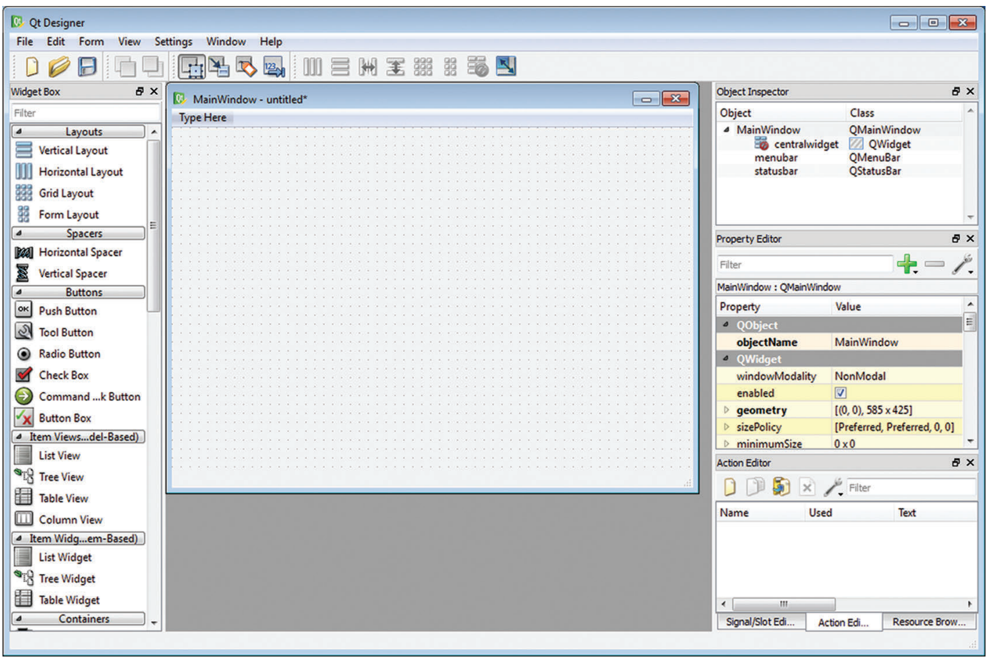
使用 Qt 时，可以使用 Qt Designer 来创建 UI。我们来看一下它的工作方式。

Qt Designer

当你安装完 PyQt 时，同时安装的还有 Qt Designer。找到它的图标（比如在 Windows 下到开始菜单中寻找），然后启动 Qt Designer。启动之后你应该可以看到 Qt 的窗口打开，中间有一个 New Form（新窗口）对话框：



Form 是 GUI 窗口对应的编程术语。因为你要创建一个新的 GUI 窗口，所以选择 Main Window（主窗口）选项，然后点击 Create（创建）按钮。现在我们来看一下 Qt 窗口的样子。



左边是组件列表，其中是能用于 GUI 的各种图形界面元素。它们被分为几个不同的分类。

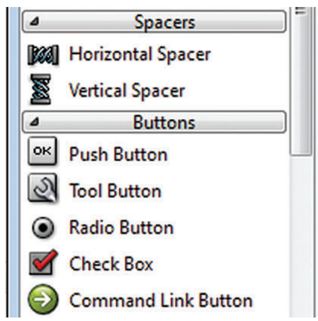
术语箱

在 GUI 中，单个的按钮、复选框等都叫做“部件”(widget)，也称为“组件”(component)，有时还称为“控件”(control)。

右边是对象检查器（Object Inspector）和属性编辑器（Property Editor），你可以在这里查看和修改组件的属性。除了对象检查器和属性编辑器，右边还有第三个面板，它的功能是由底部选中的标签决定的。它可以是 Signal/Slot 检查器、行为编辑器（Action Editor）或者资源浏览器（Resource Browser）。

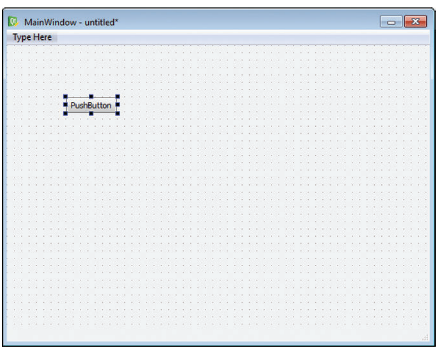
中间是刚刚创建的空窗口。最顶上显示着 Main Window - untitled，是因为你还没有给它命名。中间的空白部分就是放置组件制作 UI 界面的地方。（在 Mac 上，需要在 Qt Designer ➤ Preferences 中将 user interface mode（UI 模式）从 Multiple Top-Level Windows（浮动窗口）改为 Docked Window（集成窗口）才能看到这个界面。否则所有的面板都会浮动成独立的窗口。）

添加按钮



我们来为 GUI 添加一个按钮。在 Qt Designer 窗口的左边找到 Buttons（按钮），然后找到 Push Button 组件。

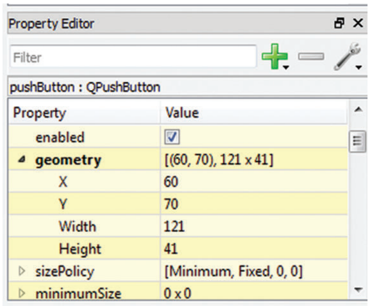
将 Push Button 拖到空白的窗口中，然后放在某个地方。现在窗口中有了一个按钮，它上面的文字是 PushButton。

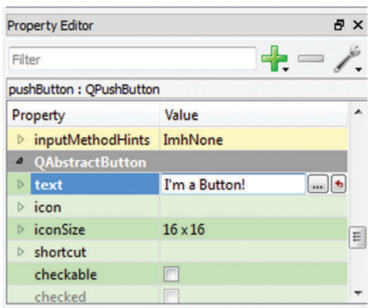


现在注意看一下右边的属性编辑器。如果按钮仍然处于选中态（周围有蓝色小方块），你会在属性编辑器中看到它的属性。可以看到按钮的名字是 PushButton。如果你在属性列表中向下滚动一下，还会看到其他的属性，比如宽度（width）、高度（height），以及 x 和 y 位置，等等。

修改按钮

要修改按钮的大小或者按钮在窗口中的位置，有两种方法：用鼠标拖动按钮，或者改变 `Size` 或 `Position` 属性。可以试着用这两种方法移动和调整按钮大小，看看它们的作用。要想使用鼠标移动按钮，点击按钮的任意部位然后拖动到新位置即





可。要想使用鼠标改变按钮的大小，则点击按钮边缘的任一蓝色方块（称为操作点，handle），然后拖动按钮的某一边或者某一角来进行缩放。如果要使用属性来改变按钮的大小，则点击 geometry 属性旁边的小三角形，它就会展开，并显示出 `X`、`Y`、`Width` 和 `Height` 属性。输入不同的值即可移动或者调整按钮大小。

你还可以改变按钮上显示的文字。现在这个文字和按钮的名称是一样的，但这不是必须的。我们来改一下，让按钮上的文字变为 I'm a Button!。在属性编辑器中，往下滚动，找到 text 属性，将它的值改为 I'm Button!。你也可以在按钮上双击，然后直接编辑文字。

现在窗口中按钮上的文字已经变成了 I'm a Button!，但该组件的名字（`objectName` 属性）仍然是 PushButton。如果你想在代码中操作按钮，则可以使用这个名字来指代该按钮。

保存 GUI

下面保存目前为止建立的 GUI。在 PyQt 中，所有 GUI 的描述都保存在一个 .ui 文件中。这个文件包含了窗口、菜单和部件的所有信息。Qt Designer 右侧的属性窗口中显示的正是同样的信息，现在我们需要把这些信息保存到一个文件中，以便 PyQt 程序运行时使用。

要保存 UI，进入 File 菜单，选择 Save As（另存为），为文件指定一个名字。下面把我们的 GUI 命名为 MyFirstGui。你会注意到，文件扩展名被自动设为了 .ui。所以这个 UI 会被保存为 MyFirstGui.ui。请确保将它保存在了你想要的文件夹中。默认情况下，Designer 会将它保存在软件本身所在的目录下，这可能不是你想要的结果。记得在点击 Save 之前先切换到你保存 Python 程序的文件夹。

可以在任何文本编辑器（包含 IDLE）中查看这个文件。如果打开这个文件，会看到这样的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>576</width>
    <height>425</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPushButton" name="pushButton">
    <property name="geometry">
     <rect>
      <x>60</x>
      <y>70</y>
      <width>121</width>
      <height>41</height>
     </rect>
    </property>
    <property name="toolTip">
     <string/>
    </property>
    <property name="text">
     <string>I'm a Button!</string>
    </property>
   </widget>
  </widget>
  <widget class="QMenuBar" name="menubar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>576</width>
     <height>21</height>
    </rect>
   </property>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
 </widget>
 <resources/>
 <connections/>
</ui>
```

定义窗口（背景）

定义按钮

看起来让人有点糊涂，不过如果再仔细看看，可以看到描述窗口的部分、描述按钮的部分，以及还没有讨论的其他部分，比如菜单和状态栏。

20.2　让 GUI 做点事情



现在有了一个非常基本的 GUI，这个窗口中包含一个按钮。不过它什么也做不了。我们还没有编写代码来告诉程序当有人点击按钮时要做些什么。这就像有一辆汽车，虽然有车身和四个轮子，但是没有发动机。尽管看起来不错，可是哪里也去不了。

我们需要一些代码让程序运行起来。对于 PyQt 程序来说，起码要有下面这些代码：

代码清单 20-1　PyQt 程序所需的最少代码

```
import sys
from PyQt4 import QtCore, QtGui, uic

form_class = uic.loadUiType("MyFirstGui.ui")[0]

class MyWindowClass(QtGui.QMainWindow, form_class):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)

app = QtGui.QApplication(sys.argv)
myWindow = MyWindowClass()
myWindow.show()
app.exec_()
```

- 导入所需的 PyQt 库
- 加载在 Designer 中创建的 UI ❶
- 为主窗口定义一个类
- 返回事件循环的 PyQt 对象
- 创建一个窗口类的实例
- 启动程序，显示 GUI 窗口

如果你想知道在第❶行结尾处的 [0] 的作用，右侧是关于它的解释。

> 在加载 UI 的那行代码结尾处添加 [0]，是因为 uic.load-UiType() 方法返回的是一个列表，其中包含两个东西：一个是 form_class，另一个是 base_class。我们只需要第一个，即 form_class，也就是列表中的 item[0]。

由 Python 就可以想见，PyQt 中的一切都是对象。每个窗口都是对象，要用 class 关键字定义。在这个程序以及所有我们要用 PyQt 编写的所有程序中，都有一个类继承自 PyQt 的 QMainWindow 类。在代码清单 20-1 中，我们调用了 MyWindow-Class 类（第 6 行），但其实这个类可以用任意名字。需要记住的是，类的定义只是一个蓝本，我们仍然需要根据蓝本去把程

序构建出来，即创建一个类的实例（instance）。我们是通过接近结尾处的 `myWindow = MyWindowClass()` 来创建实例的。`myWindow` 是 `MyWindowClass` 类的一个实例。

把上面的代码键入到 IDLE 或 SPE 编辑器窗口中，保存为 MyFirstGui.py。

- 主代码：MyFirstGui.py
- UI 文件：MyFirstGui.ui

这两个文件还要保存在同一个位置上，这样主程序才能找到 UI 文件并在程序启动时加载。

现在可以从 SPE 或 IDLE 运行这个程序。你会看到窗口打开，可以点击按钮，不过什么都不会发生。我们已经让程序运行起来，但是还没有为按钮编写任何代码。现在点击标题栏中的 ×，关闭这个程序。

下面来完成一个简单的任务。点击按钮时，让它移动到窗口中的一个新位置。增加代码清单 20-2 中的第 10 行到第 7 行代码。

代码清单 20-2 为 Hello 按钮增加一个事件处理器

```
import sys
from PyQt4 import QtCore, QtGui, uic

form_class = uic.loadUiType("MyFirstGui.ui")[0]

class MyWindowClass(QtGui.QMainWindow, form_class):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.pushButton.clicked.connect(self.button_clicked)

    def button_clicked(self):
        x = self.pushButton.x()
        y = self.pushButton.y()
        x += 50
        y += 50
        self.pushButton.move(x, y)

app = QtGui.QApplication(sys.argv)
myWindow = MyWindowClass()
myWindow.show()
app.exec_()
```

将事件处理器与事件相连接

增加这几行代码，每次点击鼠标时让按钮移动

事件处理器

点击时移动按钮

一定要让整个 `def` 块比 `class` 语句多缩进 4 个空格，如代码清单所示。为什么要这么做？这是因为所有组件都在窗口中，也就是说要作为窗口的一部分。所以按钮事件处理器的代码应该放在这个类定义内。

试着运行这个代码，看看会发生什么。下一节将详细分析这个代码。

20.3 事件处理器的返回

通过前几章的 Pygame 程序，我们已经学习了事件处理器，另外了解了如何使用事件处理器查找键盘和鼠标活动（也就是事件）。这些内容对 PyQt 同样适用。

在 MyWindowClass 中，我们定义了窗口的事件处理器。因为按钮在主窗口中，所以事件处理器也要放在这里。

首先需要告诉主窗口，我们正在为一个特定的组件编写一个事件处理器。也就是代码清单 20-2 中的第 10 行：

```
self.pushButton.clicked.connect(self.button_clicked)
```

这里我们将这个事件（self.pushButton.clicked）和它的事件处理器（self.button_clicked）进行了连接（或者绑定）。事件处理器 button_clicked 的定义从第 12 行开始。clicked 是我们能从按钮中获得的事件之一，其他的事件还包括 pressed 和 released。

像（Python）程序员一样思考

将按钮的事件和事件处理器连接起来的过程称为事件绑定。这是一个将事务连接在一起的编程术语。在 PyQt 和很多其他的事件驱动编程体系中，你会经常听到绑定某某东西的说法。通常你会将一个事件或者其他的信号量（signal）与用来处理事件或者信号量的代码进行绑定。信号量是一个编程术语，通常用于将信息从代码的一个地方传递到另外一个地方。



什么是 self

在 button_clicked() 事件处理器中，有个参数 self。第 14 章刚开始讨论对象时曾经说过，self 指示调用方法的实例。在这里，所有事件都来自背景或主窗口，

所以就是由这个窗口对象调用事件处理器。在这里，`self` 指示主窗口。你可能以为 `self` 指示所点击的组件，不过事实并不是这样；它指示的是包含组件的窗口。

20.4 移动按钮

如果想对这个按钮做些操作，怎么指示按钮呢？PyQt 会自动处理好窗口中所有组件的状态。因为你知道 `self` 指向窗口，pushButton 是组件的名字，所以可以使用 `self.pushButton` 来访问这个组件。

在代码清单 20-2 的例子中，每次点击按钮时会让它移动。按钮在窗口中的位置由它的 `geometry` 属性确定，该属性包含 x 位置、y 位置、宽度和高度。要改变这些属性，有两种方法。一种方法是使用 `SetGeometry()` 方法来改变几何属性。另一种方法（代码清单 20-2 中采用的方法）是使用 `move()` 方法，它只改变 x 位置和 y 位置，不改变宽度和高度。x 位置是与窗口左边的距离，y 位置是与窗口顶边的距离。窗口左上角的位置是 [0, 0]（这与 Pygame 中一样）。

运行这个程序时，你会看到，点击几次后，按钮会从窗口右下角消失。如果还想看到按钮，可以调整窗口的大小（拖动窗口边界或者窗口的某个角），让窗口更大，这样你就又能看到按钮了。完成时可以关闭窗口，你可以点击标题栏中的 ×，或者使用操作系统中相应的组件。

可以注意到，与 Pygame 不同，现在我们不用操心把按钮从它的老位置“擦除”，再在新位置上重绘。我们只需要移动按钮，所有这些擦除和重绘工作都会由 PyQt 来完成。

20.5 更多有用的 GUI

我们的第一个 PyQt GUI 对于了解如何在 PyQt 中建立一个 GUI 确实很不错，但是这个程序没什么用处，也没有太大意思。所以，在本章后面和第 22 章中，我们打算再完成两个项目，首先是一个小项目，另一个项目稍微大一些，通过这两个项目，我们会对 PyQt 的使用有更多了解。

第一个项目是 PyQt 版本的温度转换程序。在第 22 章中，我们将会使用 PyQt 建立游戏 Hangman 的 GUI 版本。本书后面会再次使用 PyQt 来创建电子宠物程序。

20.6 TempGUI

你已经在第 3 章“动手试一试”部分中建立了第一个温度转换程序。第 5 章中，我们又为它增加了用户输入，这样一来，需要转换的温度就不必硬编码写在程序中

了。在第 6 章中，我们使用了 EasyGui 来得到输入并且显示输出。现在我们要使用 PyQt 来建立这个温度转换程序的一个图形化版本。

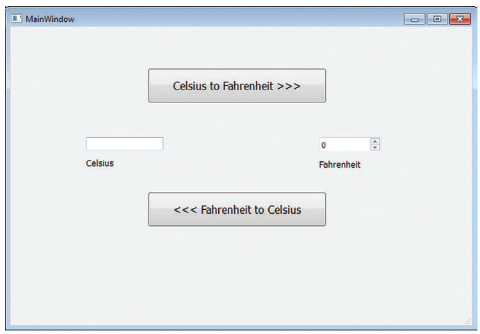
TempGUI 组件

我们的温度转换 GUI 相当简单，只需要提供以下内容：

- 输入温度的位置（摄氏度和华氏度）；
- 完成温度转换的按钮；
- 向用户显示有关信息的一些标签。

只是为了好玩，下面对摄氏度和华氏度分别使用两种不同类型的输入部件。在实际程序中绝对不要这么做（这只会把人们搞糊涂），不过这里我们的目的是学习如何使用这些部件！

建立了 GUI 布局后，会得到类似右图这样的结果：



也许你自己就可以完成，因为 Qt Designer 非常友好，很容易使用。不过没准你会需要一些帮助，所以这里还是会对相应步骤做些解释。这样也可以确保我们对组件使用相同的名字，便于更好地理解后面的代码。

并不要求组件完全对齐，也不必完全按这样来摆放组件，只要大致相同就可以了。

创建新的 GUI

第一步是建立一个新的 PyQt 工程。当你关闭打开的 UI（MyFirstGui）时，Designer 会重新打开 New Form 窗口，在确保选中 Main Window 的情况下，点击 Create 即可。

下面开始增加组件：摄氏度输入框是一个 Line Edit 组件，华氏度输入框是一个 Spin Box，各个温度输入框下面的标签都是 Label 组件，另外还有两个 push Button 组件。在 Widget Box 的左侧向下滚动，以找到这些组件。建立这个 GUI 的步骤如下。

(1) 在 Widget Box 中找到 Push Button 组件，将它拖到窗口上，窗口上会出现一个新的按钮，然后：

- 通过拖动操作点或者在 `geometry` 属性中输入新的数字（在 MyFirstGui 中见过的），将按钮改成你想要的大小。

- 将按钮的 `objectName` 属性改为 `btnFtoC`。
- 将按钮的 `text` 属性改为 <<<`Fahrenheit to Celsius`。
- 将按钮的 `font size` 改为 12。如果你在属性编辑器中找到了 `font` 属性，点击上面有三个点的按钮，它长得像这样：。然后你会看到字体对话框，你应该在自己最喜欢的文字处理软件中见过这个对话框。

(2) 向窗口中再拖入一个 Push Button，将它放到第一个按钮上方，并将尺寸改为你想要的大小，然后修改以下设置：

- 将按钮的 `objectName` 属性改为 `btnCtoF`。
- 将按钮的 `text` 属性改为 `Celsius to Fahrenheit` >>>。
- 将按钮的 `font size` 改为 12。

(3) 向窗口中拖入一个 Line Edit 组件，然后将它放到两个按钮的左边：

- 将组件的 `objectName` 属性改为 `editCel`。

(4) 向窗口中拖入一个 Spin Box 组件，然后将它放在两个按钮的右边：

- 将组件的 `objectName` 属性改为 `spinFahr`。

(5) 向窗口中拖入一个 Label 组件，然后将它放在 Line Edit 组件的下方：

- 将组件的 `text` 属性改为 `Celsius`。
- 将组件的 `font size` 改为 10。

(6) 向窗口中拖入一个 Label 组件，然后将它放在 Spin Box 的下方：

- 将挂件的 `text` 属性改为 `Fahrenheit`。
- 将挂件的 `font size` 改为 10。

现在所有 GUI 元素（组件，也称为控件或部件）都已经摆放好，并且赋予了我们想要的名字和标签。把这个 UI 文件保存为 tempconv.ui（在 Qt Designer 中选择 File ➤ Save As）。记得将文件路径改为你通常保存 Python 程序的位置。

接下来，在 IDLE 中新建一个文件，键入基本的 PyQt 代码（或者也可以从我们的第一个程序复制）：

```
import sys
from PyQt4 import QtCore, QtGui, uic

form_class = uic.loadUiType("tempconv.ui")[0]

class MyWindowClass(QtGui.QMainWindow, form_class):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
```

```
app = QtGui.QApplication(sys.argv)
myWindow = MyWindowClass()
myWindow.show()
app.exec_()
```

摄氏度转换为华氏度

首先来完成摄氏度到华氏度的转换。将摄氏度转换为华氏度的公式是：

```
fahr = cel * 9.0 / 5 + 32
```

我们需要从 editCel Line Edit 组件得到摄氏度，完成计算，再把结果放在 spinFahr 华氏度 Spin Box 组件中。这些应当在用户点击 Celsius to Fahrenheit 按钮时发生，所以要把完成这些工作的代码放在 Celsius to Fahrenheit 按钮的事件处理器中：

首先，我们需要将按钮的 `clicked` 事件连接到事件处理器：

```
self.btn_CtoF.clicked.connect(self.btn_CtoF_clicked)
```

像我们在第一个程序中所做的那样，将这段代码放在 `MyWindow` 类的 `__init__()` 方法中。

然后我们需要定义事件处理器。我们使用 `self.editCel.text()` 来从 Celsius 输入框（名为 editCel 的 Line Edit 组件）中取值。取到的值为字符串，所以我们要将它转换为浮点数：

```
cel = float(self.editCel.text()
```

然后我们来做单位转换：

```
fahr = cel * 9.0 / 5 + 32
```

接下来，我们需要将这个值放到 Fahrenheit 输入框中，也就是名为 spinFahr 的 Spin Box 组件。这里需要注意一点：Spin Box 组件的值只能是整数，不能是浮点数。所以在放入 Spin Box 之前需要将值转换为 `int`。Spin Box 的数字就是它的 `value` 属性，所以代码是这样的：

```
self.spinFahr.setValue(int(fahr))
```

我们要给结果加上 0.5，这样在使用 `int()` 将浮点数转换为整数时，才会用四舍五入以确保得到最近的整数值，而不是向下取整。将这些放到一起就是这样的：

```
def btn_CtoF_clicked(self):
        cel = float(self.editCel.text())            ← 获取摄氏度值
        fahr = cel * 9.0 / 5 + 32                   ← 转换为华氏温度
        self.spinFahr.setValue(int(fahr + 0.5))     ← 四舍五入并放入华氏温度 Spin Box
app = QtGui.QApplication(sys.argv)
myWindow = MyWindowClass()
myWindow.show()
app.exec_()
```

华氏度转换为摄氏度

要完成另一个方向的转换（从华氏度转换为摄氏度），代码很类似。这个转换的公式是：

```
cel = (fahr - 32) * 5.0 / 9
```

这个代码要放在 Fahrenheit to Celsius 按钮的事件处理器中。我们将事件处理器连接到按钮上（在窗口的 `__init__()` 方法中）：

```
self.btn_FtoC.clicked.connect(self.btn_FtoC_clicked)
```

然后，我们需要在事件处理器中从 Spin Box 获取华氏温度：

```
fahr = self.spinFahr.value()
```

这个值已经是一个整数，所以我们不必做任何类型的转换。然后应用公式：

```
cel = (fahr - 32) * 5.0 / 9
```

最后，把它转换为一个字符串，放在摄氏度文本框中：

```
self.editCel.setText(str(cel))
```

整个程序见代码清单 20-3。

代码清单 20-3 温度转换程序

```
import sys
from PyQt4 import QtCore, QtGui, uic

form_class = uic.loadUiType("tempconv.ui")[0]          ← 加载 UI 定义
class MyWindowClass(QtGui.QMainWindow, form_class):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_CtoF.clicked.connect(self.btn_CtoF_clicked)  | 绑定按钮的事
        self.btn_FtoC.clicked.connect(self.btn_FtoC_clicked)  | 件处理器

    def btn_CtoF_clicked(self):                      | CtoF 按钮的
        cel = float(self.editCel.text())             | 事件处理器
        fahr = cel * 9 / 5.0 + 32                    |
        self.spinFahr.setValue(int(fahr + 0.5))      |

    def btn_FtoC_clicked(self):                      | FtoC 按钮的
        fahr = self.spinFahr.value()                 | 事件处理器
        cel = (fahr - 32) * 5 / 9.0                  |
        self.editCel.setText(str(cel))               |

app = QtGui.QApplication(sys.argv)
myWindow = MyWindowClass(None)
myWindow.show()
app.exec_()
```

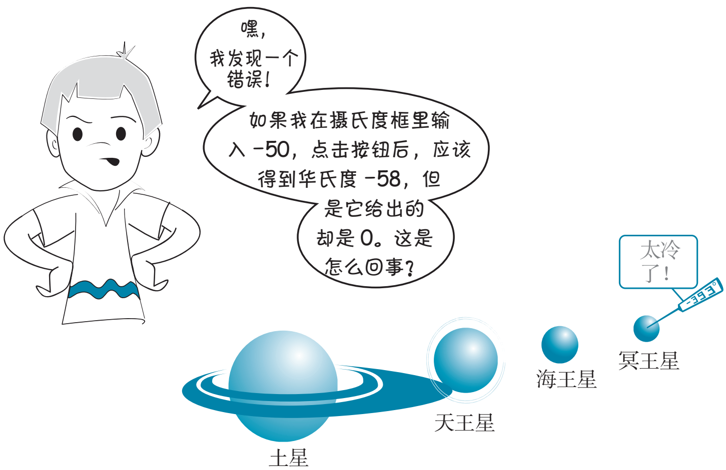
把这个程序保存为 TempGui.py。可以运行程序，试试这个 GUI。

小改进

运行这个程序时，你可能会注意到一点：将华氏度转换为摄氏度时，结果里有很多小数位，应该可以在文本框中去掉其中一些小数位。要解决这个问题有一个办法，称为打印格式化（print formatting）。我们还没有讨论到这个内容，你可以直接跳到第 21 章，那里对打印格式化的工作给出了一个完备的解释，或者也可以先直接键入这里给出的代码。用下面两行代码替换 btn_FtoC_clicked 事件处理器的最后一行。

```
cel_text = '%.2f' % cel
self.editCel.setText(cel_text)
```

这样显示一个数时会带两位小数。



嗯……也许该调试了（debug）。如果用户想转换南极洲的温度会怎么样呢？转换冥王星上的温度呢？

消灭错误

前面我们说过，要看程序中发生了什么，有一种很好的方法，就是在程序运行时打印出一些变量的值。下面就来试试看。

看起来是摄氏度到华氏度转换中的华氏度值有问题，所以就从这里开始。把下面这行代码增加到代码清单 20-3 中 btn_CtoF_clicked 事件处理器的最后一行之后：

```
print 'cel = ', cel, ' fahr = ', fahr
```

现在，一旦点击 Celsius to Fahrenheit 按钮，可以看到 IDLE shell 窗口中会打印出 `cel` 和 `fahr` 变量的值。对 `cel` 取几个不同的值，看看会发生什么。我得到了下面的结果：

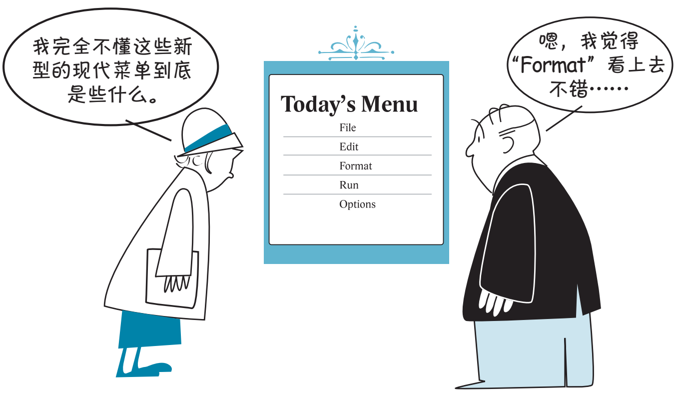
```
>>> ============================ RESTART ============================
>>>
cel =  50.0   fahr =  122.0
cel =  0.0   fahr =  32.0
cel =  -10.0   fahr =  14.0
cel =  -50.0   fahr =  -58.0
```

看起来 `fahr` 值计算得很正确。那为什么华氏度框不能显示小于 0（或大于 99）的数呢？

再回到 Qt Designer，点击用来显示和输入华氏度的 spinFahr 微调框。现在来看属性编辑器窗口，滚动属性编辑器查看不同的属性。有没有看到两个名为 `minimum` 和 `maximum` 的属性（位于微调框属性列表底端附近）？它们的值是什么？现在你能不能猜出问题出在哪里？

20.7 菜单上是什么

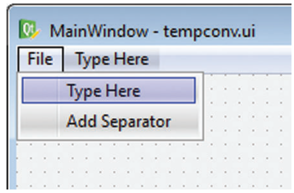
我们的温度转换 GUI 上有一些按钮，用来完成转换。很多程序还会提供一个菜单来完成某些功能。有时这些工作也可以通过点击一个按钮来完成，那为什么要采用两种不同方法完成同样的事情呢？



是这样的，有些用户更习惯使用菜单，而不喜欢点击按钮。复杂的程序可能会有很多功能，如果不使用菜单就会需要很多的按钮，这会使 GUI 变得杂乱无章。另外，菜单还可以通过键盘来操作，有些人发现，与把手从键盘拿开再使用鼠标相比，直接使用菜单速度会更快。

下面来增加一些菜单项，为用户提供另外一种完成温度转换的途径。我们也可以添加 File ➤ Exit（文件➤退出），几乎每个程序都有这个菜单项。

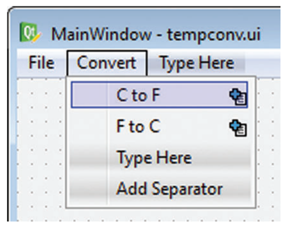
PyQt 内置了一种创建和编辑菜单的方式。如果你在 Designer 的左上角找一下，会看到 Type Here（在此输入），这就是创建菜单的地方。在很多程序中，第一个菜单项都是 File（文件）菜单，所以我们也从这个菜单开始。点击写有 Type Here 的区域，输入 File，然后按回车（Enter）。你应该可以看到 File 菜单出现了，同时在其旁边和下面还有可以输入更多菜单的区域，如右图所示。



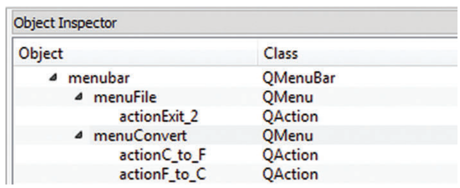
增加菜单项

在 File 菜单下，我们要添加一个 Exit（退出）菜单项。在 File 下方写有 Type Here 的区域输入 Exit，然后按回车。

现在我们来添加一个菜单项用于转换温度（假如用户不想使用按钮的话）。在 File 菜单右侧写有 Type Here 的区域，输入 Convert，然后在下方分别创建两个菜单项：C to F（摄氏转华氏）和 F to C（华氏转摄氏）。完成之后的样子应该像右图这样：



如果你看一下 Qt Designer 窗口右上角的对象查看器（Object Inspector），会看到这样的东西（如右图所示）：



你可以看到 File 和 Convert（转换）菜单，以及 Exit、C to F 和 F to C 菜单项。在 PyQt 的术语中，菜单项是 `QAction` 类的实例。这样命名是有含义的，因为你希望在选中菜单项的时候有一些“行为”（action）发生。

将你修改过的 Designer 文件保存为 tempconv_menu.ui。

现在你有了菜单项（或者说行为），你需要将它们的事件绑定（或连接）到事件处理器。对于 C to F 和 F to C 菜单项来说，我们已经有了事件处理器，即我们之前为按钮编写的事件处理器。我们希望点击菜单项时发生的事情和点击按钮一样。所以只需要将菜单项和之前的事件处理器连接起来即可。

> 在 Mac 上，你还需要将菜单栏对象的 nativeMenuBar 属性勾选掉。（它应该是属性编辑器中的最后一个属性。）否则菜单会与 Python 的主菜单冲突，PyQt 程序的 File（文件）菜单会消失。

对于菜单项（行为）来说，我们希望处理的不是 clicked 事件，而是一个 triggered 事件。我们需要连接到事件处理器的菜单项叫做 action_CtoF，而我们要连接的事件处理器则是按钮的事件处理器，叫做 btn_CtoF_clicked。将菜单项和事件处理器连接起来的代码是这样的：

```
self.action_CtoF.triggered.connect(self.btn_CtoF_clicked)
```

同样，还有一个 FtoC 菜单项。

对于 Exit 菜单项而言，我们需要创建一个新的事件处理器来绑定事件。我们将事件处理器命名为 menuExit_selected，连接事件的代码是这样的：

```
self.actionExit.triggered.connect(self.menuExit_selected)
```

Exit 菜单项的事件处理器的函数体其实只有一行代码，将窗口关闭即可：

```
def menuExit_selected(self):
self.close()
```

最后，将当前加载的 UI 文件（第三行）改为添加菜单之后保存的文件，即 tempconv_menu.ui。

做完上面这些修改之后，整个代码应该如下面的代码清单所示。

代码清单 20-4　带菜单的完整的温度转换程序

```
import sys
from PyQt4 import QtCore, QtGui, uic

form_class = uic.loadUiType("tempconv_menu.ui")[0]

class MyWindowClass(QtGui.QMainWindow, form_class):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_CtoF.clicked.connect(self.btn_CtoF_clicked)
        self.btn_FtoC.clicked.connect(self.btn_FtoC_clicked)
        self.action_CtoF.triggered.connect(self.btn_CtoF_clicked)
        self.action_FtoC.triggered.connect(self.btn_FtoC_clicked)
        self.actionExit.triggered.connect(self.menuExit_selected)

    def btn_CtoF_clicked(self):
        cel = float(self.editCel.text())
        fahr = cel * 9 / 5.0 + 32
        self.spinFahr.setValue(int(fahr + 0.5))

    def btn_FtoC_clicked(self):
        fahr = self.spinFahr.value()
        cel = (fahr - 32) * 5 / 9.0
```

加载包含菜单的 UI 文件

连接 Convert 菜单项和事件处理器

连接 Exit 菜单项和事件处理器

```
        self.editCel.setText(str(cel))
    def menuExit_selected(self):
        self.close()

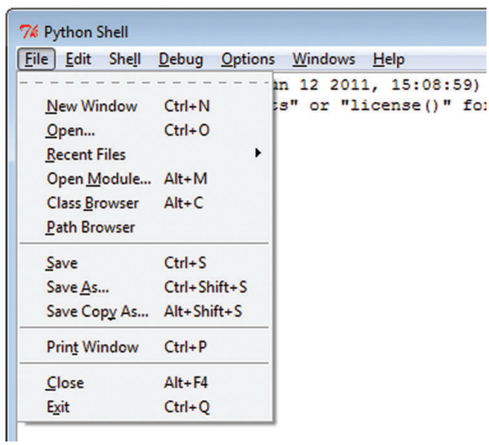
app = QtGui.QApplication(sys.argv)
myWindow = MyWindowClass(None)
myWindow.show()
app.exec_()
```

Exit 菜单项的事件处理器

热键

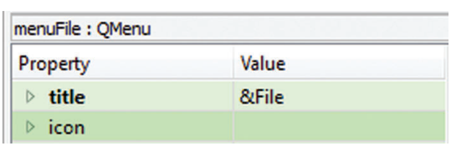
我们在前面说过，有些人喜欢用菜单而不喜欢用按钮，一个原因是菜单可以在不使用鼠标的情况下直接使用键盘操作。现在，我们的菜单已经可以用鼠标来操作了，但是还不能用键盘来操作。我们需要为菜单添加热键。

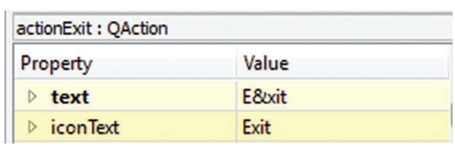
热键（也叫快捷键）让你可以使用键盘来选中菜单项。在 Windows 和 Linux 下，通过 Alt 键来激活菜单系统。（我们会马上讲到 Mac OS。）当按下 Alt 键时，你会看到各个菜单项中有个字母变成高亮显示了，通常会有一条下划线。划线的字母即是可以用来激活菜单的键。比如，要进入 File 菜单，就按 Alt-F。也就是说，先按住 Alt 键，再按 F 键，即可看到 File 菜单中的各项，同时也可以看到每个菜单项的热键是什么。在 Python Shell 窗口中试一下（见右图）。



要打开一个新窗口，则使用 Alt-F-N。（按下 Alt 键，按 F 键，再按 N 键。）

现在，我们来为温度转换 GUI 程序定义菜单的热键。要定义热键，只需在想让其成为热键的字母前加上 & 字符即可。你可以在菜单（比如 File）的 Title 属性或者菜单项（比如 Exit）的 Text 属性中完成。对于 File 菜单来说，一般使用 F 作为热键；对于 Exit 菜单项来说，一般使用 X 作为热键。所以将 File 改为 &File，将 Exit 改为 E&xit。





我们需要为 Convert 菜单项指定热键。我们为 Convert 使用 C，Celsius to Fahren-

heit 使用 C，华氏转摄氏使用 F 吧。所以我们将对应的文本分别改为 &Convert、&C to F 和 &F to C。热键组合分别是 Alt-C-C 和 Alt-C-F。

在 Qt Designer 中定义完热键之后，就不需要再做其他事情了，即没有新的代码需要编写。PyQt 和操作系统会处理热键的下划线和键盘输入。你只需保存 UI 文件即可，你可以将它存为一个不同的名字，比如 tempconv_menu_hotkeys.ui。如果使用了不同的名字，记得修改代码清单 20-4 中的第三行来加载这个 UI 文件：

```
form_class = uic.loadUiType( "tempconv_menu_hotkeys.ui" )[0]
```



答案是“没有”。因为所有的 Mac 电脑自诞生之日起就有鼠标（或者触摸板），所以 Mac OS 会假设你会使用鼠标来操作菜单。在 Mac OS 上，菜单项没有键盘快捷键或者热键系统。很多功能都有快捷键，而且其中有一些会和菜单项对应。但你并不能像在 Windows 上那样直接使用热键操作菜单系统。

这就是温度转换 GUI 的全部内容。在第 22 章，我们将会使用 PyQt 来实现一个 Hangman 游戏。

你学到了什么

在这一章，我们学到了以下内容。

- PyQt。
- Qt Designer。
- 构成 GUI 的组件——按钮、文本等。
- 事件处理器——让组件具体做事情。
- 菜单项和热键。

测试题

1. 构成 GUI 的按钮、文本域等元素有哪 3 个名字？
2. 要进入一个菜单，可以与 ALT 同时按下哪个字母？
3. Qt Designer 文件的文件名末尾必须加上什么？
4. 使用 PyQt 的 GUI 中可以包含哪 5 种组件类型？
5. 要让组件（如按钮）完成某项工作，它需要有一个 ________________________ ________________。
6. 菜单中使用哪个特殊字符来定义热键？
7. PyQt 中微调框的内容总是一个 __________。

动手试一试

1. 我们在第 1 章建立了一个基于文本的猜数程序，另外在第 6 章建立了这个程序的一个简单的 GUI 版本。试着使用 PyQt 建立这个猜数程序的 GUI 版本。
2. 前面出现微调框无法显示低于 0 的值的问题（卡特在代码清单 20-2 中找出了这个 bug），你发现了吗？修改微调框属性来解决这个问题。确保对范围上下界都做修改，使微调框不仅能显示非常高的温度，也能显示非常低的温度。（也许你的用户除了想转换冥王星上的温度，还想转换水星和金星上的温度呢！）

第 21 章
打印格式化与字符串

在第 1 章中（真是很早以前了），你已经学习了 print 语句。这是我们在 Python 中使用的第一个语句。我们还在第 5 章中见过可以在 print 语句末尾加一个逗号，让 Python 在同一行上打印后面的内容。（至少 Python 2 可以。Python 3 则不可以。）我们曾经利用这一点来建立 raw_input() 的提示语，不过后来我们了解到一种更好的快捷方法，可以把提示语直接放在 raw_input() 函数中。

这一章中，我们将要学习打印格式化，利用这些方法可以让程序的输出看起来与你希望的一样。我们将要了解下面的内容。

- 换行（以及什么时候换行）。
- 水平间隔（以及按列对齐）。
- 在字符串中间打印变量。
- 以整数、小数或 E 记法格式打印数字，以及设置应当有多少个小数位。

我们还会学习 Python 中处理字符串的一些内置方法，这些方法可以完成下面的工作。

- 将字符串分解为较小的部分。
- 将字符串联接在一起。
- 搜索字符串。
- 在字符串内搜索。
- 删除字符串中的某些部分。
- 改变大小写。

这些功能对于文本模式（非 GUI）程序非常有用，其中大部分功能在 GUI 和游戏程序中也同样有用武之地。在打印格式化方面 Python 还可以做很多其他工作，不过以上应该已经涵盖了程序中需要的 99% 的功能。

21.1　换行

print 语句我们已经见过很多次了。如果这个语句使用不只一次会发生什么？可以试试这个小程序：

```
print "Hi"
print "There"
```

运行这个程序时，输出将是：

```
>>> ================== RESTART ===================
>>>
Hi
There
```

为什么这两个内容分别打印在不同的行上？为什么输出不是这样：

```
HiThere
```

除非你另外指出，否则 Python 每次执行 print 时都会在新的一行上开始。打印 Hi 之后，Python 会下移一行，并回到第一列来打印 There。Python 会在两个词之间插入一个换行符（newline）。换行符的作用相当于在文本编辑器中按下了回车。



像程序员一样思考

还记得吧？我们在第 5 章已经了解到，CR 和 LF（回车和换行）会标志一个文本行的结束。另外我还说过，有些系统可能只使用其中一个字符（CR 或 LF）表示换行，有些系统则两个都用。换行是所有系统上行末标记的通用名。在 Windows 中，换行 = CR + LF。在 Linux 中，换行 =LF，而在 Mac OS X 中，换行 = CR。所以不必担心你使用哪个系统，希望换行时只需要加入一个换行符。

print 和逗号

print 语句会自动在它打印的内容末尾加一个换行符，除非你明确指出不要这么做。怎么告诉它不换行呢？可以加一个逗号（就像第 5 章中的一样）：

```
print 'Hi',
print 'There'

>>> ================== RESTART ===================
>>>
Hi There
```

（同样，这段代码不能在 Python 3 中正常运行。）注意 Hi 和 There 之间有一个空格。使用逗号不让 Python 打印换行符时，它会打印一个空格。

如果希望连续打印两个内容而且中间没有空格，可以使用拼接（concatenation），这在前面已经见过：

```
print 'Hi' + 'There'

>>> ==================== RESTART ====================
>>>
HiThere
```

记住，拼接就像把字符串加在一起，之所以用这个特殊的叫法是因为“相加”只适用于数字。

增加自己的换行符

如果想增加自己的换行符呢？例如，如果希望 Hi 和 There 之间空一行，该怎么做呢？最容易的办法是直接增加一个 print 语句：

```
print "Hi"
print
print "There"
```

运行这个代码时，会得到右面的结果：

```
>>> ================== RESTART ==================
>>>
Hi

There
```

特殊打印代码

增加换行符还有一种方法。Python 提供了一些特殊的代码，可以把这些代码增加到字符串中，以不同的方式打印。这些特殊的打印代码都以一个反斜线（\）字符开头。

换行符的相应代码是 \n。可以在交互模式中试一下：

```
>>> print "Hello World"
Hello World
>>> print "Hello \nWorld"
Hello
World
```

\n 使 Hello 和 World 分别打印在不同的行上，因为它在这两个词之间增加了一个换行符。

21.2 水平间隔——制表符

我们刚才看到了如何控制垂直间隔（通过增加换行，或者使用逗号来避免换

行）。现在我们来看如何利用制表符控制屏幕上内容的水平间隔。

制表符（Tab，也叫做进格符）在按列对齐方面非常有用。要了解制表符是如何工作的，可以想一想屏幕上的每一行都划分为多个大小相同的块时是什么样。下面假设每一个块为 8 个字符宽。插入一个制表符时，就会移到下一个块开始的位置。

要了解具体怎么做，最好的办法就是试一试。制表符的特殊代码是 \t，所以可以在交互模式先试试：

```
>>> print 'ABC\tXYZ'
ABC     XYZ
```

注意 XYZ 与 ABC 有几个字符的间隔。实际上，XYZ 距离这一行的起始位置正好是 8 个字符。这是因为块的大小是 8。也可以这样讲：每 8 个字符之后有一个制表点（tab stop）。

这个例子中执行了不同的 print 语句，这里增加了一些阴影来显示制表点在哪里：

```
>>> print 'ABC\tXYZ'
ABC     XYZ

>>> print 'ABCDE\tXYZ'
ABCDE   XYZ

>>> print 'ABCDEF\tXYZ'
ABCDEF  XYZ

>>> print 'ABCDEFG\tXYZ'
ABCDEFG XYZ

>>> print 'ABCDEFGHI\tXYZ'
ABCDEFGHI       XYZ
```

可以将屏幕（或者每一行）视为按 8 个空格为一块来摆放。注意，尽管 ABC 序列越来越长，但 XYZ 仍保持在原来的位置上。\t 告诉 Python 让 XYZ 从下一个制表点开始，或者从下一个可用的块开始。不过，一旦 ABC 序列长到将第一块填满，Python 就会把 XYZ 下移到下一个制表点。

按列组织内容时，制表符很有用，能让所有内容都对齐。下面就要利用这一点以及我们了解的关于循环的知识，打印一个关于正方形和立方体的表格。在 IDLE 中打开一个新窗口，键入代码清单 21-1 中的小程序，保存这个程序并运行。（我把这个程序命名为 squbes.py，这是“squares and cubes”的简写。）

代码清单 21-1 打印正方形和立方体的程序

```
print "Number \tSquare \tCube"
for i in range (1, 11):
    print i, '\t', i**2, '\t', i**3
```

运行这个程序时，应该能看到输出像下面显示的那样准确地对齐：

```
>>> ===================== RESTART ======================
>>>
Number   Square   Cube
1        1        1
2        4        8
3        9        27
4        16       64
5        25       125
6        36       216
7        49       343
8        64       512
9        81       729
10       100      1000
```

如何打印反斜线

由于反斜线字符（\）用来表示特殊打印代码，如果我们确实想打印一个 \ 字符，而不是将其作为代码的一部分打印，该如何告诉 Python 呢？我们的技巧是把两个反斜线放在一起：

```
>>> print 'hi\\there'
hi\there
```

第一个 \ 告诉 Python 接下来是一些特殊的内容，第二个 \ 告诉 Python 这些特殊的内容就是 \ 字符。

术语箱

当你使用两个反斜线来打印一个反斜线字符时，第一个反斜线叫做“转义字符”（escape character）。我们说第一个反斜线会将第二个反斜线“转义”，这样第二个反斜线就会被当成普通字符输出，而不是当成特殊字符。

21.3　在字符串中插入变量

之前，如果我们想在字符串中间加变量，都是这样做的：

```
name = 'Warren Sande'
print 'My name is', name, 'and I wrote this book.'
```

运行这个代码时，会得到：

```
My name is Warren Sande and I wrote this book.
```

不过要在字符串中插入变量还有一种方法，利用这种方法，可以更好地控制变量（特别是数字）的显示。我们可以使用格式字符串（format string），其中使用了百分号（%）。下面假设希望在 print 语句中间插入一个字符串变量，就像前面一样。

如果利用格式字符串，可以这样做：

```
name  = 'Warren Sande'
print 'My name is %s and I wrote this book' % name
```

这里有两处用到 % 符号。先是用在字符串中间，指示要把变量放在什么位置。然后在字符串后面再次用到，告诉 Python 接下来就是我们希望在字符串中插入的变量。

%s 表示我们想插入一个字符串变量。如果想插入整数，要使用 %i；想插入浮点数，则要使用 %f。

下面再给几个例子：

```
age = 13
print 'I am %i years old.' % age
```

运行这个代码时，会得到下面的输出：

```
I am 13 years old.
```

再看这个例子：

```
average = 75.6
print 'The average on our math test was %f percent.' % average
```

运行这个代码时，会得到下面的输出：

```
The average on our math test was 75.600000 percent.
```

%s、%f 和 %i 都称为格式字符串，这些代码用来指示如何显示变量。

格式字符串中还可以增加一些其他内容，从而完全按你希望的方式打印数字。你还可以使用一些不同的格式字符串得到类似 E 记法的结果。（应该还记得第 3 章介绍的 E 记法吧？）我们将在后面几节学习这些内容。

21.4 数字格式化

打印数字时，我们希望对数字如何显示有一些控制：

- ❒ 显示多少位小数；
- ❒ 使用常规记法还是 E 记法；
- ❒ 是否增加前导或末尾的 0；
- ❒ 是否在数字前面显示正负号（+ 或 -）。

利用格式字符串，Python 为我们提供了充分的灵活性，不仅可以完成这些工作，甚至还可以做更多事情！

例如，如果你在使用一个天气预报程序，你想看到下面哪一个结果呢？是这样：

```
Today’s High: 72.45672132, Low 45.4985756
```

还是这样：

```
Today’s High: 72, Low: 45
```

适当地显示数字对很多程序来说都很重要。

下面先来看一个例子。假设我们想要打印一个有两位小数的浮点数。试着在交互模式中执行以下命令：

```
>>> dec_number = 12.3456
>>> print 'It is %.2f degrees today.' % dec_number
It is 12.35 degrees today
```

`print` 语句中间包含一个格式字符串。不过这一次没有直接使用 `%f`，而是使用了 `%.2f`。这就告诉 Python 要采用浮点数格式，而且小数点后面要显示两位。（注意，Python 非常聪明，它会正确地把数字四舍五入为两位小数，而不是直接去掉多余的数位。）



这个字符串后面，第二个 `%` 字符告诉 Python 接下来就是要打印的数。这个数要采用格式字符串中描述的格式来打印。再看几个例子就能明白了。

你的记性不错，卡特！`%` 符号确实用作取余操作符（整数除法中求余数），这是我们在第 3 章中学过的，不过它也用于指示格式字符串。Python 能够区分出你是指取余还是格式字符串。

整数：%d 或 %i

要把某项内容打印成整数，可以使用 `%d` 或 `%i` 格式字符串。（我不知道为什么会有两个，不过用哪个都可以。）

```
>>> number = 12.67
>>> print '%i' % number
12
```

注意，这一次数字没有四舍五入。它被截断（truncated，表示“直接切断”）了。如果是四舍五入，我们会看到 13 而不是 12。使用整数格式化时，数字会被截断，不过使用浮点数格式化时，数字则会四舍五入。

这里要注意以下 3 个方面。

- 字符串中不要求有其他文本，可以只包含格式字符串。
- 即使是浮点数，也可以把它打印为整数。这可以通过格式字符串实现。

- Python 会把值截断为上一个最大整数[①]。不过，这与 int() 函数（在第 4 章见过）不同，因为格式字符串不会像 int() 那样创建一个新的值，而只是改变值显示的方式。

刚刚我们用整数格式打印 12.67，结果 Python 打印出了 12。不过变量 number 的值并没有改变。可以检查一下：

```
>>> print number
12.67
```

number 的值并没有改变。我们只是使用格式字符串采用不同的方式打印。

浮点数：%f 或 %F

打印小数时，可以在格式字符串中使用大写或小写的 f（%f 或 %F）：

```
>>> number = 12.3456
>>> print '%f' % number
12.345600
```

如果只使用 %f，数字会显示为有 6 位小数。如果在 f 前面加上 .n，这里 n 可以是任意整数，就会把数字四舍五入为指定的小数位数：

```
>>> print '%.2F' % number
12.35
```

可以看到它把数字 12.3456 四舍五入到小数点后前两位：12.35。

如果指定的小数位比数中实际的小数位还要多，Python 会用 0 来填充（pad）：

```
>>> print '%.8f' % number
12.34560000
```

这个数的小数点后面只有 4 位，但我们要求有 8 位小数，所以另外 4 位会用 0 来填充。

如果是负数，%f 总会显示 – 号。如果希望数字总会显示正负号（即使它是一个正数），可以在 % 后面加一个 + 号（如果列表中既有正数也有负数，这对于列表的对齐也很有好处）：

```
>>> print '%+f' % number
+12.345600
```

如果希望包含正负数的列表对齐，但是不希望看到正数带 + 号，可以在 % 后面

① 原文这里是“下一个最小整数”，这是不对的。例如，13.2 会截断为 13，也就是上一个最大整数，而下一个最小整数应当是 14，13.2 当然不会截断为 14。——译者注

使用一个空格代替 +：

```
>>> number2 = -98.76
>>> print '% .2f' % number2
-98.76
>>> print '% .2f' % number
 12.35
```

输出中的 12 前面有一个空格，所以，尽管 98 前面有负号而 12 前面没有正负号，这两个数也能对齐。

E 记法：%e 和 %E

我们在第 3 章讨论过 E 记法，前面说过我会告诉你如何使用 E 记法来打印数字。好吧，现在就来介绍这个内容。

```
>>> number = 12.3456
>>> print '%e' % number
1.234560e+01
```

要使用 %e 格式字符串打印 E 记法。它总是打印 6 位小数，除非你另作要求。

如果要打印更多或更少的小数位，可以在 % 后面使用 .n，就像打印浮点数时一样：

```
>>> number = 12.3456
>>> print '%.3e' % number
1.235e+01
>>> print '%.8e' % number
1.23456000e+01
```

%.3e 四舍五入为 3 位小数，%.8e 增加了一些 0 来填充不足的小数位。

小写或大写 e 都是可以的，你在格式字符串中使用什么大小写形式，输出时也会显示同样的大小写：

```
>>> print '%E' % number
1.234560E+01
```

自动浮点数或 E 记法：%g 和 %G

如果你希望 Python 自动选择浮点数记法或 E 记法，可以使用 %g 格式字符串。同样，如果使用大写，输出中就会得到一个大写的 E：

```
>>> number1 = 12.3
>>> number2 = 456712345.6
>>> print '%g' % number1
12.3
>>> print '%g' % number2
4.56712e+08
```

注意到了吗？ Python 会为大数自动选择 E 记法，而对较小的数使用浮点数记法。

如何打印百分号

你可能很想知道，既然百分号（%）对格式字符串来说是一个特殊的字符，那么怎么打印 % 呢？

嗯，Python 有时候很聪明，它能确定你什么时候使用 % 符号开始一个格式字符串，以及什么时候只是想打印一个百分号。可以试试这个命令：

```
>>> print 'I got 90% on my math test!'
I got 90% on my math test!
```

在这个例子中，字符串外面没有第二个 %，也没有需要格式化的变量，所以 Python 认为这个 % 只是字符串中的一个普通字符。

但是如果你在打印格式化字符串时想打印一个百分号，就需要输入两个百分号，就像你曾经用两个反斜线来打印一个反斜线一样。我们说第一个百分号对第二个百分号进行了转义，就像在本章前面的术语箱中提到的一样：

```
>>> math = 75.4
>>> print 'I got %.1f%% on my math test' % math
I got 75.4% on my math test
```

第一个 % 表示格式化字符串。两个 %% 在一起告诉 Python，你希望打印出一个 % 字符。引号外面的 % 告诉 Python 你需要将后面的变量打印出来。

多个格式化字符串

如果你想在一个 print 语句中放入多个格式化字符串，该怎么办呢？你可以这样做：

```
>>> math = 75.4
>>> science = 82.1
>>> print 'I got %.1f in math and %.1f in science' % (math, science)
```

你可以在 print 语句中放入任意多个格式化字符串，后面跟一个变量元组（tuple）。还记得吗？元组和列表很像，只是它使用的是圆括号而不是方括号，而且元组是不可变的。这是 Python 显得很挑剔的一个地方，这里你必须使用元组，而不能使用列表。除非只有一个变量需要格式化，这时才可以不用元组。（你在很多例子中已经见过了。）请确保（引号内的）格式化字符串的数量和（引号外的）变量数量相同，否则程序会报错。

存储格式化数字

有时你不想直接打印出格式化的数字，而是希望把它存储在一个字符串中以备以后使用。这很容易，可以不打印，把它直接赋给一个变量，如下：

```
>>> my_string = '%.2f' % 12.3456
>>> print my_string
12.35
>>> print 'The answer is', my_string
The answer is 12.35
```

这里没有直接打印格式化数字，我们把它赋给变量 `my_string`。再将 `my_string` 与其他一些文本结合，并打印整个句子。

对于 GUI 和其他图形程序（如游戏）来说，将格式化数字存储为字符串非常有用。一旦有一个对应格式化字符串的变量名，就可以采用你希望的任何方式来显示：可以显示在文本框中，显示在对话框中，或者显示在游戏屏幕上。

21.5 格式化的新方法

刚刚学习的格式化字符串的语法在 Python 的所有版本中都可以正常工作。但是在 Python 2.6 及之后的版本中，有一个新的格式化方法。因为这本书中使用的是 Python 2.7，所以我想我们也应该看一下这个新方法。你可能会在 Python 代码中见到它，所以这样你起码会知道它是什么意思。你可以自行决定使用新的还是旧的语法来格式化字符串。

format() 方法

（在 Python 2.6 及之后的版本中）Python 字符串有一个名为 `format()` 的方法。它的工作方式和你在前面见过的 `%` 格式化字符串很像。事实上，格式化说明符号 `f`、`g`、`e` 等都是一样的，只不过使用方式稍有区别。最好用例子来说明。

下面是旧的方法：

```
print 'I got %.1f in math, %.1f in science' % (math, science)
```

下面是新的方法：

```
print 'I got {0:.1f} in math, {1:.1f} in science'.format(math, science)
```

在这种新方法中，格式化说明符被放在花括号中，而不再以 `%` 开头。`0` 和 `1` 告诉 Python 要格式化的是元组中的哪个变量。Python 计数从 0 开始，所以元组中第一个元素（变量 `math`）的索引是 0，第二个元素（变量 `science`）的索引是 1。然后使用 `.1f`，其用法和旧方法中一样。

这就是新方法的全部内容。你可以像旧方法中用 % 格式化一样，将格式化后的字符串存为一个变量：

```
distance = 149597870700
myString = 'The sun is {0:.4e} meters from the earth'.format(distance)
```

因为你不再使用 % 来区分格式化字符串，所以如果要打印一个 % 的话，也不需要再做任何特殊工作：

```
>>> print 'I got {0:.1f}% in math'.format(math)
I got 87% in math
```

Python 程序员们会觉得使用 format() 语法更好，尤其是在 Python 3 中，但你可以自由选择。本书中的例子都使用 % 语法。

21.6　更多字符串处理

最早学习字符串时（第 2 章），我们已经看到，可以用 + 号把两个字符串联接起来，就像这样：

```
>>> print 'cat' + 'dog'
catdog
```

现在来看还可以对字符串做哪些处理。

Python 中的字符串实际上都是对象（看到了吧，所有一切都是对象……），而且有自己的方法来完成搜索、分解和结合之类的操作。这些方法都称为字符串方法。刚刚看到的 format() 方法就是一种字符串方法。

分解字符串

有时需要把一个长字符串分解成多个小字符串。通常你会想在字符串的某些特定位置（比如说出现某个字符的地方）进行分解。例如，在文本文件中存储数据时，常见的方法就是将各个项用逗号分隔。所以你可能会得到类似这样一个名字字符串：

```
>>> name_string = 'Sam,Brad,Alex,Cameron,Toby,Gwen,Jenn,Connor'
```

假设你想把这些名字放在一个列表中，每一项是一个名字。这就需要在出现逗号的地方分解字符串。完成这项工作的 Python 方法名为 split()，它的用法如下：

```
>>> names = name_string.split(',')
```

要指出使用哪个字符作为分解标记，这个方法会返回一个列表，也就是把原来的字符串分解为许多部分。如果打印这个例子的输出，这个长长的名字串会分解为一个列表中的单个列表项：

```
>>> print names
['Sam','Brad','Alex','Cameron','Toby','Gwen','Jenn','Connor']

>>> for name in names:
        print name

Sam
Brad
Alex
Cameron
Toby
Gwen
Jenn
Connor
```



也可以用多个字符作为分解标记。例如，可以使用 'Toby,' 作为分解标记，这会得到下面的列表：

```
>>> parts = name_string.split('Toby,')
>>> print parts
['Sam,Brad,Alex,Cameron', 'Gwen,Jenn,Connor']

>>> for part in parts:
        print part

Sam,Brad,Alex,Cameron
Gwen,Jenn,Connor
```

这一次，字符串会分解为两部分：'Toby,' 左侧的所有内容，以及 'Toby,' 右侧的所有内容。注意 'Toby,' 并没有出现在列表中，因为分解标记会被丢掉。

还有一点要知道。如果没有为 Python 指定任何分解标记，它会按空白符（whitespace）分解字符串：

```
>>> names = name_string.split()
```

空白符表示所有空格、制表符或换行符。

联接字符串

我们刚才了解了如何把一个字符串分解为较小的部分。那么怎样把两个或多个字符串联接起来构成一个较长的字符串呢？（在第 2 章中）我们已经了解到，可以使用 + 操作符把字符串联接起来。这就像把两个字符串相加，只不过这称为拼接（concatenating）。

联接字符串还有一种方法。可以使用 join() 函数。你要指出你希望把哪些字符串联接起来，另外希望在联接的各部分之间插入什么字符（如果有的话）。这实际上与 split() 正相反。下面是交互模式中完成的一个例子：

```
>>> word_list = ['My', 'name', 'is', 'Warren']
>>> long_string = ' '.join(word_list)
>>> long_string
'My name is Warren'
```

我得承认这看起来有些怪异。要联接的各个字符串之间可以插入字符，而且这个字符居然放在 join() 前面。在这里，我们希望每个词之间有一个空格，所以使用了 ''.join()。大多数人可能都没有想到会是这样，不过 Python 的 join() 方法确实要这样使用。

下面这个例子会让人觉得我是只小狗：

```
>>> long_string = ' WOOF WOOF '.join(word_list)
>>> long_string
'My WOOF WOOF name WOOF WOOF is WOOF WOOF Warren'
```

换句话说，join() 前面的字符串可以用作粘合剂，把其他字符串联接在一起。

搜索字符串

假设你想为妈妈创建一个程序，获取食谱并在 GUI 中显示。你想在一个位置显示配料，在另一个位置显示做法。假设食谱是这样的。

```
Chocolate Cake
Ingredients:
2 eggs
1/2 cup flour
1 tsp baking soda
1 lb chocolate

Instructions:
Preheat oven to 350F
Mix all ingredients together
Bake for 30 minutes
```

假设食谱中的各行都被放在一个列表中，每一行在列表中都是单独的元素。怎么找到“Instructions”（做法）部分呢？Python 提供了两种有用的方法。

startswith() 方法可以指出一个字符串是否以某个字符或某几个字符开头。举个例子最能说明问题。在交互模式中试试右面的例子。

```
>>> name = "Frankenstein"
>>> name.startswith('F')
True
>>> name.startswith("Frank")
True
>>> name.startswith("Flop")
False
```

名字 Frankenstein 确实以字母 F 开头，所以第一个结果是 True。名字 Frankenstein 确实以 Frank 开头，所以第二个结果也是 True。不过，名字 Frankenstein 不是以 Flop 开头，所以这一个结果是 False。



因为 startswith() 方法返回一个 True 或 False 值，所以可以在比较或 if 语句中使用这个方法，比如可以这样：

```
>>> if name.startswith("Frank"):
        print "Can I call you Frank?"
```

还有一个类似的方法，名为 endswith()，从这个方法名你应该可以想到它会做什么。

```
>>> name = "Frankenstein"
>>> name.endswith('n')
True
>>> name.endswith('stein')
True
>>> name.endswith('stone')
False
```

现在，回到我们的问题……如果想找到食谱的“Instructions”部分从哪里开始，可以这样做：

```
i = 0
while not lines[i].startswith("Instructions"):
    i = i + 1
```

这个代码会一直循环，直到找到以“Instructions”开头的一行。应该记得，lines[i] 表示 i 是 lines 的索引。所以要从 lines[0]（第 1 行）开始，然后是 lines[1]（第 2 行），依此类推。while 循环结束时，i 会等于“Instructions”开头的那一行的索引，这正是你要找的位置。

在字符串中搜索：in 和 index()

startswith() 和 endswith() 方法可以很好地查找位于字符串开头和末尾位置的内容。不过如果想在一个字符串中间找某个内容呢？

下面假设你有一些包含街道地址的字符串，如下：

```
657 Maple Lane
47 Birch Street
95 Maple Drive
```

也许你想找出所有包含 Maple 的地址。这里所有字符串都不是以 Maple 开头或结尾的，但是其中有两个确实包含有单词 Maple。怎么找到它们呢？

实际上，我们已经知道怎么做了。前面讨论列表时（第 12 章），曾经见过可以用这种方法来检查某个元素是否在一个列表中：

```
if someItem in my_list:
    print "Found it!"
```

这里使用了关键字 in 来检查某个元素是否在列表中。关键字 in 也同样适用于字符串。实际上字符串就是一个字符列表，所以可以这样做：

```
>>> addr1 = '657 Maple Lane'
>>> if 'Maple' in addr1:
        print "That address has 'Maple' in it."
```

术语箱

在较大的字符串（如 657 Maple Lane）中查找较小的字符串（如 Maple）时，较小的这个字符串称为子串（substring）。

in 关键字只能指出子串是不是位于你检查的字符串中的某个位置，但没有告诉你它到底在什么位置。要得到这个位置，需要使用 index() 方法。类似于搜索列表，index() 会指出较小串从较大字符串中的哪个位置开始。右面是一个例子。

```
>>> addr1 = '657 Maple Lane'
>>> if 'Maple' in addr1:
        position = addr1.index('Maple')
        print "found 'Maple' at index", position
```

如果运行这个代码，会得到右面的输出：

```
found 'Maple' at index 4
```

单词 Maple 从字符串 657 Maple Lane 的位置 4 开始。与列表一样，字符串中字母的索引（或位置）都是从 0 开始，所以 M 位于索引 4。

6	5	7		M	a	p	l	e		L	a	n	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13

注意，使用 index() 之前，我们首先用 in 查看子串 Maple 是不是确实在较大的字符串中。这是因为，如果使用了 index()，而你查找的内容不在字符串中，就会得到一条错误消息。先用 in 检查可以杜绝这样的错误。在第 12 章中我们对列表也是这样做的。

删除字符串的一部分

你可能常常希望删除或剥除字符串的某一部分。通常，你希望剥除末尾部分，

如换行符或一些多余的空格。Python 提供了一个字符串方法 strip()，完全可以做到这一点。只需要告诉它你想剥除哪一部分，如下：

```
>>> name = 'Warren Sande'
>>> short_name = name.strip('de')
>>> short_name
'Warren San'
```

在这里剥除了我名字末尾的 de。如果末尾根本没有 de，那么什么也不会剥除：

```
>>> name = 'Bart Simpson'
>>> short_name = name.strip('de')
>>> short_name
'Bart Simpson'
```

如果没有告诉 strip() 要剥除哪一部分，它就会去除所有空白符。前面说过，这包括空格、制表符和换行符。所以，如果想要去除一些多余的空格，就可以这样做：

```
>>> name = "Warren Sande     "
>>> short_name = name.strip()
>>> short_name
'Warren Sande'
```

注意我名字末尾的多余空格

注意我名字后面多余的空格都已经删除。这里有一点很好：你不需要告诉 strip() 要删除多少个空格，它会删除字符串末尾的所有空白符。

改变大小写

我还要告诉你两种字符串方法。可以使用这两种方法把字符串从大写转换为小写，或者反过来，从小写转换为大写。有时你可能希望比较两个字符串，比如 Hello 和 hello，你想知道它们包含的字母是不是相同（尽管大小写可能不完全一样）。一种办法是让两个字符串中的所有字母都变成小写，然后完成比较。

Python 为此提供了一个字符串方法，名为 lower()。可以在交互模式中试试下面的命令：

```
>>> string1 = "Hello"
>>> string2 = string1.lower()
>>> print string2
hello
```

还有一个类似的方法，名为 upper()：

```
>>> string3 = string1.upper()
>>> print string3
HELLO
```

可以为原来的字符串建立全小写（或全大写）的副本，然后比较这两个副本，看看它们是否相同（忽略大小写）。

你学到了什么

在这一章，你学到了以下内容。

- 如何调整垂直间隔（添加或删除换行符）。
- 如何用制表符设置水平间隔。
- 如果使用格式字符串显示不同的数字格式。
- 使用格式字符串的两种方法：`%` 符号和 `format()` 方法。
- 如何用 `split()` 分解字符串和用 `join()` 联接字符串。
- 如何使用 `startswith()`、`endswith()`、`in` 和 `index()`。
- 如何用 `strip()` 去除字符串末尾的部分。
- 如何用 `upper()` 和 `lower()` 将字符串转换为全大写或全小写。

测试题

1. 如果有两个单独的 `print` 语句，如下：

```
print "What is"
print "your name?"
```

   怎样把所有内容都打印在同一行上？
2. 打印时如何增加额外的空行？
3. 实现按列对齐时要使用哪一个特殊打印代码？
4. 使用哪个格式字符串强制按 E 记法打印一个数？

动手试一试

1. 编写一个程序，询问一个人的姓名、年龄和最喜欢的颜色，然后打印在一句话里。运行这个程序时应该看到类似这样的结果：

```
>>> ======================== RESTART ========================
>>>
What is your name? Sam
How old are you? 12
What is your favorite color? green
Your name is Sam you are 12 years old and you like green
```

2. 还记得第 8 章中的乘法表程序（代码清单 8-5）吗？现在编写这个程序的一个改进版本，使用制表符确保所有内容都能很好地按列对齐。
3. 编写一个程序计算 8 的所有分数（例如，1/8, 2/8, 3/8……直到 8/8），要显示 3 位小数。

第 22 章

文件输入与输出

你是不是很想知道，你喜欢的计算机游戏为什么能记住高分，甚至计算机关机之后还能记得？你的浏览器又怎么能记住你喜欢的网站呢？这一章我们就来学习这是怎么做到的。

前面已经说过多次，程序包括 3 个主要方面：输入、处理和输出。到目前为止，输入主要直接来自用户，也就是从键盘和鼠标输入；输出都直接发送到屏幕。（如果是声音就会发送到扬声器。）不过，有时我们还需要使用其他来源的输入。通常程序需要使用存储在某个地方的输入，而不是在程序运行时才由用户输入。有些程序需要从计算机硬盘上的文件得到输入。

例如，如果建立一个 Hangman 游戏，你的程序需要一个单词表，可以从中选择秘密词。这个单词表必须存储在某个地方，可能是随程序提供的“单词表”文件。程序要打开这个文件，读取单词表，并选择一个词在程序中使用。

输出也一样。有时需要把程序的输出存储起来。程序使用的所有变量都是临时的，也就是说，程序一旦停止运行，这些变量就会丢失。如果想保存某些信息以便以后使用，就必须把它们存储在可以永久保存的地方，比如说存储在硬盘中。例如，如果想维护某个游戏的高分表，要把这些高分存储在一个文件中，这样下次程序运行时，就可以读取这个文件并显示这些分数。

在本章，我们将了解如何打开文件以及如何读写文件（从文件获取信息和在文件中存储信息）。

22.1 什么是文件

在讨论打开和读写文件之前，我们先来看看什么是文件。

前面说过，计算机按二进制格式存储信息，二进制只使用 1 和 0。每个 1 或 0 称

为一位（bit），8 位一组称为一个字节（byte）。文件是有名字的字节集合，存储在硬盘、CD、DVD、软盘、flash 盘或其他存储介质上。

文件可以存储很多不同类型的信息。一个文件可以包含文本、图片、音乐、计算机程序、电话号码表等内容。计算机硬盘上的所有内容都以文件的形式存储。程序就是由一个或多个文件构成的。你的计算机的操作系统（比如 Windows、Mac OS X 或 Linux）需要很多很多文件才能运行起来。

文件有以下属性：

- 名字
- 类型，表明文件中包含什么类型的数据（图片、音乐、文本）
- 位置（文件存储在哪里）
- 大小（文件中有多少字节）

22.2　文件名

大多数操作系统中（包括 Windows），文件名中有一部分用来指示文件中包含什么类型的数据。文件名中通常至少有一个点（.），点后面的部分指出了文件的类型。这一部分称为扩展名（extension）。

来看下面这几个例子。

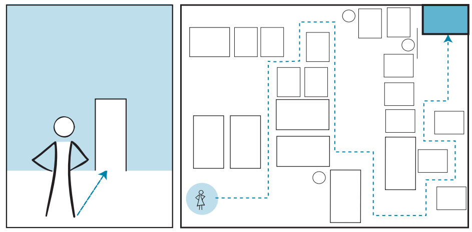
- my_letter.txt 中的扩展名是 .txt，代表“文本”，所以这个文件可能包含文本。
- 在 my_song.mp3 中，扩展名是 .mp3，这是一种声音文件。
- 在 my_program.exe 中，扩展名是 .exe，这代表“可执行文件”。在第 1 章我曾经提到过，“执行”就是指运行一个程序，这只是“运行程序”的另一种说法。所以 .exe 文件往往是可以运行的程序。
- 在 my_cool_game.py 中，扩展名是 .py，通常表示一个 Python 程序。

Mac OS X 中，程序文件（文件中包含一个可以运行的程序）扩展名是 .app，代表“应用”，这是“程序”的另一种说法。

有一点很重要，你可以根据自己的喜好给文件命名，还可以使用任何扩展名。例如，你可以建立一个文本文件（例如，在记事本程序 Notepad 中建立），但命名为 my_notes.mp3。这并没有把它变成一个声音文件，这个文件中仍然只包含文本，所以这实际上是一个文本文件。你只是给了它一个特别的文件扩展名，让它看上去像是一个声音文件，这可能会让人莫名其妙，也会把计算机搞得稀里糊涂。给文件命名时，最好使用一个与文件类型一致的扩展名。

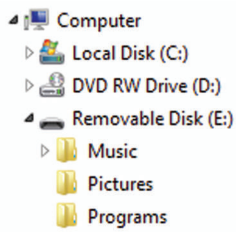
22.3 文件位置

到目前为止，我们一直在处理与程序存储在相同位置上的文件。我们没有考虑如何查找文件，因为它与程序在同一个地方。

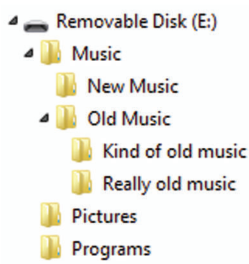


这就像你在自己的房间里时，你不用担心找不到你的壁橱，它就在房间里。但是如果你在另一个房间、另一幢房子或者在另一个城市里，要找到壁橱就复杂多了！

每个文件都要存储在某个地方，所以除了文件名外，每个文件还有自己的位置。硬盘和其他存储介质都组织为文件夹或目录。文件夹（folder）和目录（directorie）表示的是同一样东西，只是名字不同而已。它们是一种组织文件的方法。文件夹或目录组织和关联的方式称为文件夹结构或目录结构。



在 Windows 中，每个存储介质由一个字母表示，如 C 代表硬盘，E 对应一个闪存盘。在 Mac OS X 和 Linux 上，每个存储介质都有一个名字（例如，hda 或 FLASH DRIVE）。每个存储单元可以划分为多个文件夹，如 Music、Pictures 和 Programs。如果查看文件浏览器（如 Windows Explorer），就像右图这样：



文件夹中还可以有其他文件夹，这些文件夹本身又可以包含另外的文件夹，依此类推。右边这个例子包含了 3 层文件夹：

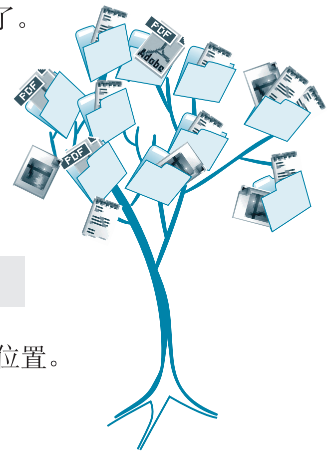
第一层是 Music，下一层包含 New Music 和 Old Music，再下一层包含 Kind of old music 和 Really old music。

术语箱

位于其他文件夹中的文件夹称为子文件夹（subfolder）。如果使用术语“目录”来描述，可以把它们称为子目录（subdirectory）。

在 Windows Explorer（或其他文件浏览器）中查找文件或文件夹时，文件夹就像一棵树的分支。“根”是驱动器本身，如 C: 或 E:。每个主文件夹就像树干，各个主文件夹中的文件夹则像小树枝，依此类推。

不过，从程序访问文件时，这种树型想法就不适用了。你的程序不能点击文件夹，不能通过浏览整棵树来查找某个文件，它需要一种更直接的方法来查找文件。好在还有另外一种方法可以表示树结构。点击不同文件夹和子文件夹时，如果你查看 Windows Explorer 的地址栏，你会看到这样的地址：




```
E:\Music\Old Music\Really old music\my_song.mp3
```

这称为路径（path），描述了文件在文件夹结构中的位置。

这个特定的路径表达的意思如下：

(1) 从 E: 盘开始；
(2) 进入名为 Music 的文件夹；
(3) 在 Music 文件夹中，进入一个名为 Old Music 的子文件夹；
(4) 在 Old Music 子文件夹中，进入下一层一个名为 Really old music 的子文件夹；
(5) 在 Really old music 子文件夹中，有一个名为 my_song.mp3 的文件。

可以使用类似这样的路径找到计算机上的任何文件。程序就是利用这种方法来查找和打开文件的。下面是一个例子：

```
image_file = "c:/program files/HelloWorld/examples/beachball.png"
```

使用文件的完全路径名总能找到文件。完全路径名包含从根（驱动器，如 C:）开始这个路径上的所有文件夹名。这个例子中的文件名就是一个完全路径名。

斜线还是反斜线

斜线（\ 和 /）一定要正确使用，这很重要。Windows 在路径名中可以接受斜线 (/) 也可以接受反斜线（\），不过如果在 Python 程序中使用类似 `c:\test_results.txt` 的路径，\t 部分会带来问题。还记得吗？在第 21 章中，我们谈到过一些用于打印格式化的特殊字符，如 \t 表示制表符。正是因为这个原因，所以应当避免在文件路径中出现 \ 字符。Python（和 Windows）会把 \t 看作是一个制表符，而不是像你预想的那样把它当作文件名的一部分。所以应当使用 /。

另一种选择是使用双反斜线，如下：

```
image_file  "c:\\program files\\HelloWorld\\images\\beachball.png"
```

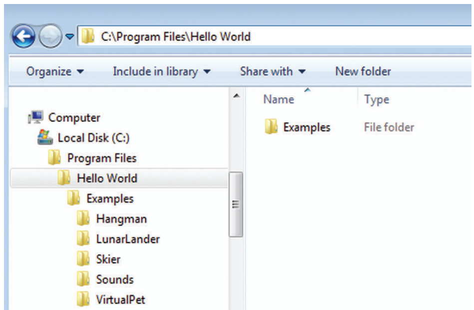
记住，如果希望打印一个 \ 符号，必须在它前面再放一个反斜线。在文件名中也是如此。不过我还是推荐使用 /。

有时并不需要完整的文件路径。下一节将讨论如何在“半路上”查找一个文件。

看看你在哪里

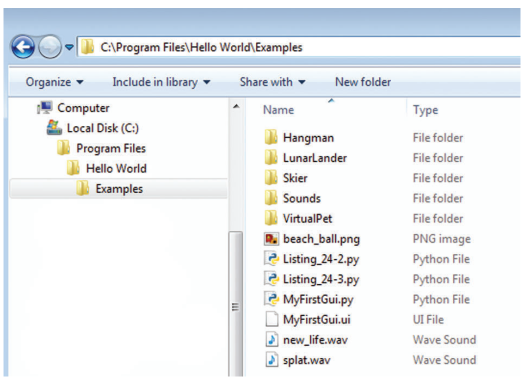
大多数操作系统（包括 Windows）都有一个“工作目录”概念，有时也称为“当前工作目录”，这是文件夹树中你目前所在的目录。

假设从根（C:）开始，沿着 Program Files 分支向下移到 Hello World 分支。你的当前位置或当前目录就是 C:/Program Files/Hello World。



现在要找到文件 beachball.png，必须沿 Examples 分支向下。所以达到这个文件的路径就是 /Examples/beachball.png。由于你已经在这条路上走了一段，所以只需要走完剩下的一段就能到达你想去的地方。

还记得吗？在第 19 章讲到关于声音的内容时，我们打开声音文件使用的是 splat.wav 之类的文件名，并没有使用路径。这是因为，那时我告诉你要把声音文件复制到保存程序的同一个文件夹中。如果在 Windows Explorer 中查看，就是这样：



注意，我把 Python 文件（扩展名为 .py）与声音文件（扩展名为 .wav）放在同一个文件夹中。运行 Python 程序时，它的工作目录就是存储 .py 文件的文件夹。

如果把程序存储在 e:/programs 并运行，这个程序就会把 e:/programs 作为它的工作目录开始运行。如果有一个声音文件存储在同一个文件夹中，那么程序只需要它的文件名就可以打开这个声音文件。并不需要一个路径来找到这个文件，因为文件已经在当前位置上了，所以可以直接这样写：

```
my_sound = pygame.mixer.Sound("splat.wav")
```

注意，我们不需要使用声音文件的完全路径名（它的完全路径名是 e:/programs/splat.wav）。这里直接使用了文件名而没有带路径，因为这个文件与使用该文件的程序在同一个文件夹中。

关于路径已经讲得够多了

路径和文件位置就讲到这里。关于文件夹和目录、路径、工作目录等的话题让有的人感觉很迷糊，需要大量篇幅才能解释清楚。不过本书讨论的是编程，而不是操作系统、文件位置或路径，所以如果你在这个方面遇到困难，可以让你的爸爸妈妈、老师或者懂计算机的人来帮你。

本书中所有其他使用文件的例子都将文件放在与程序相同的位置，所以我们不必担心路径或使用完整路径名的问题。

22.4　打开文件

打开文件之前，需要知道你要对文件做些什么：

- 如果你要使用这个文件作为输入（只查看文件中有什么，而不做任何改变），就是要打开文件完成读；
- 如果要创建一个全新的文件或者用某个全新的文件替换现有的文件，就是要打开文件完成写；
- 如果要为一个现有文件增加内容，就是要打开文件完成追加。（记得在第 12 章我们曾经说过追加就表示做出补充吧。）

打开一个文件时，要在 Python 中建立一个文件对象。（看到了吧，我说过 Python 中的很多东西都是对象。）建立文件对象要使用 `open()` 函数，并提供文件名，就像这样：

```
my_file = open('my_filename.txt','r')
```

文件名是一个字符串（string），所以两边需要加引号。`'r'` 部分表示我们要打开这个文件来完成读。下一节还会学习更多相关内容。

一定要了解文件对象和文件名之间的区别，这很重要。我们在程序中要用文件对象来访问文件，而文件名是 Windows（以及 Linux 和 Mac OS X）对磁盘上的文件的称呼（即文件的名字）。

人也一样。我们在不同场合会使用不同的名字。如果你的老师名叫 Fred Weasley，你会叫他 Weasley 老师。他的朋友可能叫他 Fred，而他的计算机用户名可能是 fweasley。对于文件，会有一个由操作系统使用的名字，操作系统要用这个名字把文件存储在磁盘上（文件名），另外还有一个由程序使用的名字，程序处理文件时要使用这个名字（文件对象）。

这两个名字（也就是对象名和文件名）不一定要完全相同。可以把对象命名为你想使用的任何名字。例如，如果有一个包含一些说明的文本文件，名为 notes.txt，可以这样做：

```
notes = open('notes.txt', 'r')
```

File object　　文件对象　　文件名

也可以这样做：

```
some_crazy_stuff = open("notes.txt", 'r')
```

文件对象　　文件名

一旦打开文件并创建文件对象，就不再需要文件名了。我们在程序中将使用文件对象来完成所有工作。

22.5 读文件

上一节提到，可以使用 open() 函数打开文件并创建文件对象。这是 Python 的内置功能之一。要打开文件来完成读，需要使用 'r' 作为第二个参数，如下：

```
my_file = open('notes.txt', 'r')
```

如果想打开一个文件完成读，但是这个文件根本不存在，你就会得到一条错误消息。（毕竟，你无法读一个原本没有的东西，对不对？）

Python 还提供了另外一些内置功能，一旦文件打开可以将信息从文件获取到你的程序中。要从一个文件读取文本行，可以使用 readlines() 方法，如下：

```
lines = my_file.readlines()
```

这会读取整个文件，并建立一个列表，每个文本行作为列表中的一项。下面假设 notes.txt 文件包含一个小列表，上面写的都是你每天要做的事情：

```
Wash the car
Make my bed
Collect allowance
```

我们可以使用“记事本”（Notepad）之类的程序来创建这个文件。其实，你可以现在就动手，使用记事本（或者你喜欢的文本编辑器）来建立这样的文件。可以

把它命名为 notes.txt，保存在 Python 程序所在的位置，然后关闭记事本。

如果用一个小 Python 程序打开并读取这个文件，代码可能如代码清单 22-1 所示。

代码清单 22-1　打开和读文件

```
my_file = open('notes.txt', 'r')
lines = my_file.readlines()
print lines
```

输出可能是这样的（取决于你在文件中放入的内容）：

```
>>>======================== RESTART ========================
>>>
['Wash the car\n', 'Make my bed\n', 'Collect allowance']
```

这里从文件读取了文本行，并放入一个名为 `lines` 的列表中。这个列表中的每一项都是一个字符串，包含从文件读取的一行，注意前两行末尾的 \n 部分。这些是分隔文件中各行的换行符。我们创建文件时在这里按下了回车键。如果键入最后一行后按了回车键，那么在第三项后面也会有一个 \n。

代码清单 22-1 的程序中还要增加一点。处理完文件时，一定要关闭文件：

```
my_file.close()
```



嗯，卡特，倘若另一个程序需要使用这个文件，而我们的程序又还没有将它关闭，那个程序就无法访问这个文件了。使用完文件后就关闭它，这样通常会比较好。

一旦把文件读取为程序中的一个字符串列表，接下来就可以任意处理它了。这个列表与其他 Python 列表是一样的，所以可以循环处理、排序、追加元素、删除元素等等。这些字符串也像其他字符串一样，可以打印、转换为 `int` 或 `float`（如果包含数字的话）、用作 GUI 中的标签，或者完成能够对字符串做的其他处理。

一次读取一行

`readlines()` 方法会读取文件的所有行，直到文件末尾。如果你想一次只读取一行，可以使用 `readline()` 方法，如下：

```
first_line = my_file.readline()
```

这只会读文件的第一行。如果再在同一个程序中使用 readline()，Python 会记住目前在什么位置。所以，第二次使用时，你会得到文件的第二行。代码清单 22-2 显示了这样的一个例子。

代码清单 22-2 多次使用 readline()

```
my_file = open('notes.txt', 'r')
first_line = my_file.readline()
second_line = my_file.readline()
print "first line = ", first_line
print "second line = ", second_line
my_file.close()
```

这个程序的输出是这样的：

```
>>>=================== RESTART ====================
>>>
first line =  Wash the car

second line =  Make my bed
```

readline() 方法一次只读取一行，所以它不会把结果放入一个列表。每次使用 readline() 时，都只是得到一个字符串。

回到起始位置

如果已经使用了几次 readline()，现在希望退回到文件的起始位置，可以使用 seek() 方法，就像这样：

```
first_line = my_file.readline()
second_line = my_file.readline()
my_file.seek(0)
first_line_again = my_file.readline()
```

seek() 方法会让 Python 找到文件中你指示的位置。括号中的数字就是从文件起始位置算起的字节数。所以如果把它设置为 0，就会回到文件的起始位置。

22.6 文本文件和二进制文件

到目前为止，打开文件和读取文本行的所有例子都有一个假设，认为文件中实际上都包含有文本。要记住，文件能够存储任何内容，文本只是其中的一种。程序员把所有其他类型的文件都统称为二进制文件（binary file）。

可以打开的文件主要有以下两种类型。

- ❐ 文本文件：这些文件包含了文本，包括字母、数字、标点符号和一些特殊字符，如换行符。
- ❐ 二进制文件：这些文件不包含文本，它们可能包含音乐、图片或其他类型的数据。不过由于不包含文本，所以这些文件中也没有行，因为根本不存在换行符。

这说明不能对二进制文件使用 `readline()` 或 `readlines()`。例如，如果想要从一个 .wav 文件读取一“行”，你根本无法知道会得到些什么。大多数情况下，你可能会得出一大堆奇怪的东西，就像这样：

```
>>> f = open('splat.wav', 'r')
>>> print f.readline()
RIFFö▲  WAVEfmt ►   ☺ ☺ "V  "V  datap▲
ÇÇÇÇÇÇÇÇüÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ⌂ÇÇ⌂⌂⌂⌂⌂Ç⌂⌂⌂⌂Ç
ÇÇÇ⌂⌂ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ⌂Ç⌂ÇÇÇÇüÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ⌂ÇÇÇÇüüÇÇÇ⌂ÇÇ⌂ÇÇÇÇ⌂⌂ÇÇüéééÇzvvy{|Çâ
   çïê}trv|äëïîèå~ut|⌂yrqrtxÇîÖ₧æäàütvçÆÄ|mlfWR]jnmpxüêÅ ºƒâràó«¼Ö}`ORj⌂{hZZg-
   wàëy{äæá⌐¿ÿézåÿèmWLISjÇàzrvÇüytv~üÇ}yrifjt}äêèêëÄöÉémSCFZlrtyéïö¥ñ¬½⌐ñ¢ÆÄìÅôòÆ
   ÄÅæ|åÜ⌐¬ÿüpd\UME@;99:>EJMW]YTZfuçòƒ⌐▒║┴──├───┴╕⌐¢ôë~{|{yxzzuiZNG-
   HLSbs⌂~wrnf\TPQU]`jvàæÉ⌂osÇïôæä}üàë₧┼
```

.wav 文件最前面有一些东西看起来像是文本，不过后面就很莫名其妙了。这是因为 .wav 文件不包含文本，只包含声音。`readline()` 和 `readlines()` 方法只能用于读取文本文件。

大多数情况下，如果需要使用二进制文件，就要通过 Pygame 或其他一些模块来加载文件，就像在第 19 章中一样。

```
pygame.mixer.music.load('bg_music.mp3')
```

那样就会由 Pygame 负责打开文件并读取二进制数据（在这个例子中，二进制数据就是音乐）。

这本书不打算介绍如何处理二进制文件。不过没准你想知道二进制是什么样子，可以为文件模式增加一个 b 来打开二进制文件，就像这样：

```
my_music_file = open('bg_music.mp3', 'rb')
```

这里的 `'rb'` 部分表示我们要打开文件并以二进制模式读取文件。

在前面几节中，我们已经了解了如何将信息从文件读入程序，这称为读文件。接下来我们要学习如何将程序的信息放入到文件中，这称为写文件。

22.7　写文件

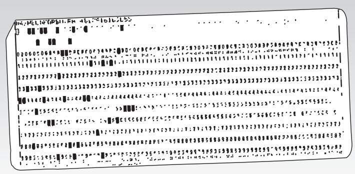
如果你想更持久地存储程序的信息，可以看着屏幕，把这些信息抄写在纸上。不过这样根本无法体现出计算机的作用！

比较好的做法是把信息保存在硬盘上，这样一来，即使程序不再运行（即计算机已经关机），你的数据仍然能保留下来，供以后使用。其实你早已经这么做了。每次保存学校作业、图片、歌曲或者 Python 程序时，实际上都是将它们存储到硬盘上。

从前的美好时光



在我们那个年代，只有纸！没有显示器、打印机，甚至没有键盘。要通过在卡片上打孔来“写”代码。然后把这叠卡片塞进一个大机器里，它会把这些打孔卡转换成计算机能够理解的电子信号。有时需要花几天时间才能得到一个答案。唉，可真是又费劲又麻烦！



老式计算机打孔卡

前面已经提到，在文件中添加内容有两种方法。

- 写——这表示开始新文件，或者覆盖现有的文件。
- 追加——这表示增加到现有的文件，保留原来已有的内容。

要写文件或追加文件，首先必须打开文件。像前面一样要使用 open() 函数，只不过第二个参数有所不同。

- 要读文件，需要使用 'r' 作为文件模式：

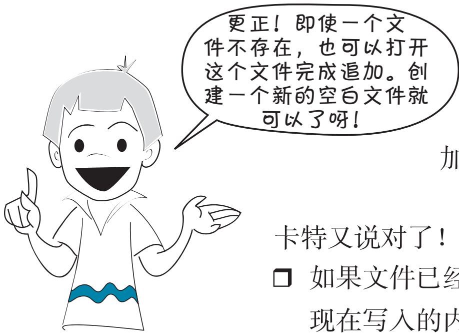
```
my_file = open('new_notes.txt', 'r')
```

- 要写文件，需要使用 'w' 作为文件模式：

```
my_file = open('new_notes.txt', 'w')
```

- 要追加文件，需要使用 'a' 作为文件模式：

```
my_file = open('notes.txt', 'a')
```



如果使用 'a' 表示追加模式，文件名必须是硬盘上某个已经存在的文件的名字，否则你会得到一条错误消息。这是因为，追加是指增加到一个现有的文件。

卡特又说对了！如果使用 'w' 表示写模式，会有两种可能：

- 如果文件已经存在，文件中的所有内容都会丢失，替换为现在写入的内容；
- 如果文件不存在，会创建一个同名的新文件，你写的所有内容会被放入这个新文件中。

下面来看一些例子。

追加到文件

首先，还是使用前面创建的 notes.txt 文件，为它追加一些内容。下面增加一行“Spend allowance”。完成 readlines() 例子时如果你仔细观察，可能已经注意到最后一行末尾没有 \n，也就是说没有换行符。所以现在需要增加一个换行符，然后再增加我们的新字符串。要把字符串写入文件，需要使用 write() 方法，如代码清单 22-3 所示。

代码清单 22-3　使用追加模式

```
todo_list = open('notes.txt', 'a')          ← 以追加模式打开文件
todo_list.write('\nSpend allowance')        ← 将字符串加到最后
todo_list.close()                           ← 关闭文件
```

读文件时，我们说过一旦完成就应当关闭文件。这一点在写文件时更为重要，写文件完成时一定要使用 close()。这是因为，只有使用 close() 关闭文件，你所做的修改才会真正保存到文件中。

运行代码清单 22-3 中的程序之后，用“记事本”（或者任何其他文本编辑器）打开 notes.txt，看看里面的内容。记住，看完后一定要关闭“记事本”。

写文件

现在来看一个使用写模式来写文件的例子。我们将打开一个目前硬盘上还没有的文件。键入代码清单 22-4 中的程序，然后运行。

代码清单 22-4　对一个新文件使用写模式

```
new_file = open("my_new_notes.txt", 'w')
new_file.write("Eat supper\n")
new_file.write("Play soccer\n")
new_file.write("Go to bed")
new_file.close()
```

怎么知道这个程序是否起作用呢？检查保存这个程序（代码清单 22-4）的文件夹，应该能看到一个名为 my_new_notes.txt 的文件。可以在“记事本”中打开这个文件，看看里面有什么。应该能看到：

```
Eat supper
Play soccer
Go to bed
```

你利用这个程序创建了一个文本文件，并在这个文件中存储了一些文本。这个文件存放在硬盘上，只要硬盘没有坏，它就会一直在那里，除非你删除了它。这样一来我们就得到了一种方法，可以持久地存储程序的数据。现在你的程序就能在这个世界上（或者至少在你的硬盘上）留下永久的印记了。如果要在程序停止和计算机关机时保留一些信息，都可以放在文件中。

下面来看如果对硬盘上已有的一个文件使用写模式会发生什么。还记得那个 notes.txt 文件吗？如果运行过代码清单 22-3 中的程序，这个文件现在是这样的：

```
Wash the car
Make my bed
Collect allowance
Spend allowance
```

下面用写模式打开这个文件，并写入内容，看看会发生什么。代码清单 22-5 给出了相应的代码。

代码清单 22-5　对一个现有文件使用写模式

```
the_file = open('notes.txt', 'w')
the_file.write("Wake up\n")
the_file.write("Watch cartoons")
the_file.close()
```

运行这个代码，然后在“记事本”中打开 notes.txt，看看其中包含什么内容。应该会看到：

```
Wake up
Watch cartoons
```

notes.txt 原先的内容都不见了，已经被代码清单 22-5 程序中的新内容所取代。

使用 `print` 写文件

上一节中，我们使用了 `write()` 来写文件，还可以用 `print` 写文件。仍然要以

写模式或追加模式打开文件，不过打开文件后可以使用 print 写文件，就像这样：

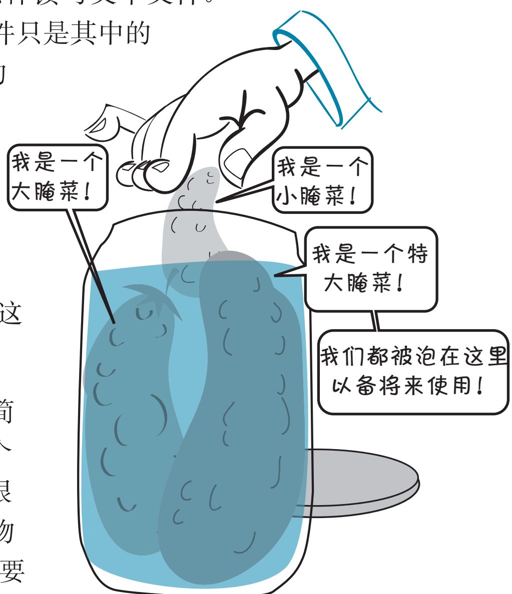
```
my_file = open("new_file.txt", 'w')
print >> my_file, "Hello there, neighbor!"
my_file.close()
```

这里的两个 > 符号（有时称为山形符号）告诉 print 要把它的输出发送到一个文件中而不是屏幕上。这称为重定向（redirecting）输出。

有时使用 print 比 write() 更方便，因为 print 还会额外完成一些工作，比如把数字自动转换为字符串等。如果要在文件中放入文本，你可以使用 print，也可以使用 write()。

22.8　在文件中保存内容：pickle

在本章第一部分中，我们讨论了怎样读写文本文件。在硬盘上存储信息有很多方法，文本文件只是其中的一种。如果你想存储列表或对象之类的内容呢？有时列表中的元素可能是字符串，不过并不一定是这样。另外，对象又该怎么存储呢？也许可以把所有对象的属性都转换为字符串，再写到一个文本文件中，但是之后你还得把这个过程反过来，从文件恢复对象。这就复杂化了。



幸运的是，Python 提供了一种更简便的方法来存储列表和对象。这是一个 Python 模块，名为 pickle。这个名字很滑稽，可以这样想：腌菜是一种储藏食物以备以后使用的方法。在 Python 中，你要把数据“腌起来”（pickle），使数据能够保存在硬盘上供以后使用。这很有道理！

使用 pickle

假设有一个列表，其中包含不同类型的内容，如下：

```
my_list  = ['Fred', 73, 'Hello there', 81.9876e-13]
```

要使用 pickle，首先必须导入 pickle 模块：

```
import pickle
```

要“腌”某个东西，比如列表，需要使用 dump() 函数。（腌菜要倒到罐子里，想到这一点就很容易记住这个函数[1]。）dump() 函数需要一个文件对象，我们知道如何建立文件对象：

```
pickle_file = open('my_pickled_list.pkl', 'w')
```

这里用 'w' 模式打开文件来完成写，因为我们要在这个文件中存储一些内容。可以选择你想要的任何文件名和扩展名。我选择 .pkl 作为扩展名，这是“pickle”的简写。

然后用 dump() 把列表“倒”在 pickle 文件中：

```
pickle.dump(my_list, pickle_file)
```

整个过程见代码清单 22-6。

代码清单 22-6　使用 pickle 将列表存储到文件中

```
import pickle
my_list  = ['Fred', 73, 'Hello there', 81.9876e-13]
pickle_file = open('my_pickled_list.pkl', 'w')
pickle.dump(my_list, pickle_file)
pickle_file.close()
```

使用这个方法可以在文件中存储任何类型的数据结构。但是怎么把它们取回来呢？这就是我们下面要谈到的内容。

还原

在现实生活中，只要把某个东西腌起来，它就一直是腌菜了。你不可能撤销这个过程，也就是说，不能把一个腌菜还原成新鲜菜。不过在 Python 中，利用 pickle“储藏”一些数据时，确实可以把这个过程反过来，取回原先的数据。

完成这种“还原”的函数是 load()。为这个函数提供一个文件对象（对应包含“被腌”数据的文件），它会按原来的格式返回数据。下面就来试试看。如果运行过代码清单 22-6 中的程序，在程序所在的相同位置上应该已经有一个名为 my_pickled_list.pkl 的文件。现在试试代码清单 22-7 中的程序，看你能不能得到原来的列表。

代码清单 22-7　使用 load() 还原

```
import pickle
pickle_file = open('my_pickled_list.pkl', 'r')
recovered_list = pickle.load(pickle_file)
pickle_file.close()

print recovered_list
```

① 函数名 dump 的含义就是“倾倒”。——译者注

应该能得到这样的输出：

```
['Fred', 73, 'Hello there', 8.1987599999999997e-012]
```

看来真还原了！我们又得到了之前“被腌”的元素。E 记法看起来有点不同，不过还是同一个数，至少 16 位小数是一样的。这里的差别是四舍五入造成的，这个问题我们在第 4 章讨论过。

在下一节中，我们将使用前面学到的文件输入和输出知识建立一个新游戏。

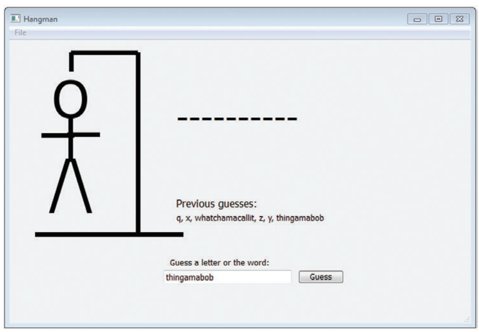
22.9　又到了游戏时间——Hangman

既然讨论的是文件，为什么要在这一章建立一个游戏呢？嗯，Hangman 游戏之所以有趣，原因在于它有一个庞大的词汇表，可以从这个词汇表中选择题目。要做到这一点，最容易的办法就是从文件中读取。我们仍然使用 PythonCard 来完成这个游戏，也想由此说明并非只能使用 Pygame 建立图形化游戏。

我不打算像介绍其他程序那样详细地解释这个程序。现在你应该已经会看代码了，相信你能自己搞清楚其中大部分代码的作用。我只会稍稍给你一点指导，帮助你顺利读懂代码。

Hangman GUI

我们的 Hangman 程序的主 GUI 是像右图这样的：



这里显示了上吊小人的各个部分，不过程序运行时，我们首先会隐藏小人的所有部分。玩家猜错一个字母时，就会显示这个小人的下一部分。如果整个小人都显示出来，玩家可以再猜一次，然后游戏结束！

玩家猜一个字母时，程序会查看这个字母是否在秘密词中。如果确实是秘密词中的字母，就把这个字母显示出来。在窗口下方，玩家可以看见到目前为止他猜过的所有字母。玩家什么时候都可以尝试猜词。

下面概括一下这个程序的工作原理。

开始时，程序完成以下工作：

- ❐ 从文件加载词汇表；
- ❐ 从每行的末尾去除换行符；

- 让小人的所有部分都不可见；
- 从词汇表随机选择一个词；
- 根据秘密词中的字母个数显示相同数目的横线。

玩家点击 Guess! 按钮时，程序做以下工作。

- 检查猜的是一个字母还是一个词。
- 如果是一个字母：
  - 检查秘密词，查看是否包含这个字母。
  - 如果玩家猜对了，用这个字母取代横线，显示这个字母出现在什么位置。
  - 如果玩家猜错了，显示小人的另一部分。
  - 把猜出的字母增加到 Previous guesses 显示区。
  - 查看玩家是否已经猜出单词（猜出所有字母）。
- 如果是一个词：
  - 检查玩家猜得正确与否。
  - 如果正确，显示一个对话框，指出 You Won!（你赢了！）并开始新游戏。
- 查看玩家是不是已经没有机会了，如果是，显示一个对话框，指出 You Lost（你失败了），并显示这个秘密词到底是什么。

从词汇表得到单词

这一章讨论的是文件，所以下面来看程序中得到词汇表的部分。相应代码如下：

```
f = open("words.txt", 'r')
self.lines = f.readlines()
for line in self.lines:
    line.strip()          从各行删除换行符
f.close()
```

words.txt 文件只是一个文本文件，所以可以使用 readlines() 读这个文件。为了从词汇表中选择一个词，我们使用了 random.choice() 函数，如下：

```
self.currentword = random.choice(self.lines)
```

显示小人

要跟踪已经显示了小人的哪些部分，另外下一步要显示哪一部分，这有很多方法。我们决定使用一个循环，代码如下：

```
def wrong(self):
        self.pieces_shown += 1
        for i in range(self.pieces_shown):
            self.pieces[i].setHidden(False)
        if self.pieces_shown == len(self.pieces):
            message = "You lose.  The word was " + self.currentword
            QtGui.QMessageBox.warning(self,"Hangman",message)
            self.new_game()
```

我们使用 `self.pieces_shown` 来跟踪上吊小人显示了多少部分。如果所有的部分都显示出来了，我们就使用一个对话框来告知玩家他输了。

检查猜到的字母

这个程序最难的一部分就是检查玩家猜到的字母，看它是否出现在秘密词中。这个工作之所以困难，是因为字母可能在一个单词中出现多次。例如，如果秘密词是 lever，玩家猜到了 e，就必须把第 2 个和第 4 个字母都显示出来，因为它们都是 e。

我们有几个函数来完成这项工作。`find_letters()` 函数会查找某个字母在单词中出现的所有位置，并返回一个包含这些位置的列表。例如，对于字母 e 和单词 lever，这个函数会返回 [1, 3]，因为字母 e 出现在这个字符串的索引 1 和索引 3 的位置上。（记住，索引从 0 开始。）代码如下：

```
def find_letters(letter, a_string):
    locations = []
    start = 0
    while a_string.find(letter, start, len(a_string)) != -1:   ← 检查字母出现的位置
        location = a_string.find(letter, start, len(a_string))
        locations.append(location)
        start = location + 1                                   | 用字母取代横线
    return locations
```

`replace_letters()` 函数从 `find_letters()` 得到列表，用正确的字母替换这些位置上的横线。在我们的例子中（lever 中的字母 e），它会用 `-e-e-` 替换 `-----`。这就向玩家显示出猜对的字母出现在单词的什么位置，其余仍然为横线。代码如下：

```
def replace_letters(string, locations, letter):
    new_string = ''
    for i in range (0, len(string)):
        if i in locations:
            new_string = new_string + letter
        else:
            new_string = new_string + string[i]
    return new_string
```

玩家猜一个字母时，我们使用刚才定义的两个函数 find_letters() 和 replace_letters()：

```
    if len(guess) == 1:
        if guess in self.currentword:
            locations = find_letters(guess, self.currentword)
            self.word.setText(replace_letters(str(self.word.text()),
                                      locations,guess))
            if str(self.word.text()) == self.currentword:
                self.win()
        else:
            self.wrong()
```

猜得是一个字母吗？

检查字母是否包含在单词中

检查字母出现在什么位置

用字母替换横线

检查是不是已经没有横线了（这说明你赢了！）

整个程序大约 95 行代码，另外我还加入了一些空行，让代码看起来更美观。代码清单 22-8 给出了整个程序，这里加了对各个不同部分做的一些解释。如果使用本书的安装程序，你的计算机上的 \Examples\Hangman 文件夹中应该已经有这个代码了，另外也可以在网站上找到这个代码，包括 hangman.py、hangman.ui 和 words.txt。切记，正如我们在第 20 章提到的，如果你使用的是 Mac，则需要在 Qt Designer 中打开 hangman.ui，并勾选掉 menubar 对象的 nativeMenuBar 属性。

代码清单 22-8 完整的 hangman.py 程序

```
import sys
from PyQt4 import QtCore, QtGui, uic
import random

form_class = uic.loadUiType("hangman.ui")[0]

def find_letters(letter, a_string):
    locations = []
    start = 0
    while a_string.find(letter, start, len(a_string)) != -1:
        location = a_string.find(letter, start, len(a_string))
        locations.append(location)
        start = location + 1
    return locations

def replace_letters(string, locations, letter):
    new_string = ''
    for i in range (0, len(string)):
        if i in locations:
            new_string = new_string + letter
        else:
            new_string = new_string + string[i]
    return new_string

def dashes(word):
    letters = "abcdefghijklmnopqrstuvwxyz"
    new_string = ''
```

查找字母

玩家猜对字母时，用字母替换横线

程序开始时用横线替换字母❶

```
    for i in word:
        if i in letters:
            new_string += "-"
        else:
            new_string += i
    return new_string

class MyWidget(QtGui.QMainWindow, form_class):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.btn_guess.clicked.connect(self.btn_guess_clicked)        连接事件
        self.actionExit.triggered.connect(self.menuExit_selected)     处理器
小人部分 self.pieces = [self.head, self.body, self.leftarm, self.leftleg,
                       self.rightarm, self.rightleg]
        self.gallows = [self.line1, self.line2, self.line3, self.line4]  绞架部分
        self.pieces_shown = 0
        self.currentword = ""
        f=open("words.txt", 'r')
        self.lines = f.readlines()          得到词汇表
        f.close()
        self.new_game()

    def new_game(self):
        self.guesses.setText("")
        self.currentword = random.choice(self.lines)     从词汇表中随机选择一个单词
        self.currentword = self.currentword.strip()
        for i in self.pieces:
            i.setFrameShadow(QtGui.QFrame.Plain)          隐藏小人
            i.setHidden(True)
        for i in self.gallows:
            i.setFrameShadow(QtGui.QFrame.Plain)
        self.word.setText(dashes(self.currentword))      调用函数，用横线替换字母
        self.pieces_shown = 0

    def btn_guess_clicked(self):                          让玩家猜字母或单词
        guess = str(self.guessBox.text())
        if str(self.guesses.text()) != "":
            self.guesses.setText(str(self.guesses.text())+", "+guess)
        else:
            self.guesses.setText(guess)
        if len(guess) == 1:                               猜字母
            if guess in self.currentword:
                locations = find_letters(guess, self.currentword)
                self.word.setText(replace_letters(str(self.word.text()),
                                                  locations,guess))
                if str(self.word.text()) == self.currentword:
                    self.win()
            else:
                self.wrong()
        else:
            if guess == self.currentword:                 猜单词
                self.win()
            else:
```

```
                self.wrong()
            self.guessBox.setText("")

    def win(self):                                              玩家猜对
        QtGui.QMessageBox.information(self,"Hangman","You win!")  时显示对
        self.new_game()                                          话框

    def wrong(self):
        self.pieces_shown += 1
        for i in range(self.pieces_shown):           显示小人的
            self.pieces[i].setHidden(False)          另一部分
        if self.pieces_shown == len(self.pieces):                       猜错
玩家输了    message = "You lose.  The word was " + self.currentword
            QtGui.QMessageBox.warning(self,"Hangman", message)
            self.new_game()
    def menuExit_selected(self):
        self.close()

app = QtGui.QApplication(sys.argv)
myapp = MyWidget(None)
myapp.show()
app.exec_()
```

为了简化起见，我们的 Hangman 程序只使用了小写字母。我们提供的单词表只有小写字母，用户也必须将其猜测的内容以小写形式输入。

新游戏开始时，❶处的 dashes() 函数使用横杠替换了字母。但它并没有替换标点符号，比如撇号。所以如果单词是 doesn't，玩家将会看到 -----'-。

建议你自己创建这个程序。可以用 Qt Designer 构建 GUI，即使看上去与这里的版本不完全一样也没有关系。不过一定要仔细查看代码，看看组件分别使用什么名字。代码中的名字必须与 .ui 文件中的名字一致。

尽可能自己键入代码。运行程序，看看结果怎么样。如果你想做些不同的尝试，那就放手去做！充分尝试，大胆试验，并享受其中的快乐。这正是编程最有意思也最有收获的地方，大多数知识都是通过这个途径获得的。

你学到了什么

在这一章，你学到了以下内容。

- ❒ 什么是文件。
- ❒ 如何打开和关闭文件。
- ❒ 打开文件的不同方式：读、写和追加。
- ❒ 写文件的不同方法：write() 或 print >>。

- 如何使用 pickle 在文件中保存列表和对象（以及其他 Python 数据结构）。
- 很多与文件夹（也称为目录）、文件位置和路径相关的内容。

我们还建立了一个 Hangman 游戏，这个游戏使用来自文件的数据得到一个词汇表。

测试题

1. Python 中用来处理文件的对象称为 _______。
2. 如何创建一个文件对象？
3. 文件对象和文件名之间有什么区别？
4. 完成文件读写时应该对文件做什么操作？
5. 如果用追加模式打开一个文件然后在文件中写入内容会怎样？
6. 如果用写模式打开一个文件然后在文件中写入内容会怎样？
7. 读过文件的一部分之后如何从文件起始位置开始读？
8. 将一个 Python 对象保存到文件中要使用哪个 pickle 函数？
9. 要“还原”一个对象 (从 pickle 文件得到对象 , 并放回到 Python 变量中)，应当使用哪个 pickle 方法？

动手试一试

1. 建立程序造一些滑稽句子。每个句子应该至少有 4 部分，类似于：

   The ________ ________ ________ ________
   （形容词）（名词）（动词短语）（副词短语）

   例如："The crazed monkey played a ukulele on the table."
   形容词 名词 动词短语 副词短语

   这个程序会随机选择一个形容词、一个名词、一个动词短语和一个副词短语来创建句子。这些单词都存储在文件中，可以使用“记事本”创建这些单词。要完成这个程序，最简单的方法是为这 4 组单词分别创建一个文件，不过也可以采用你希望的任何方式创建文件。下面给出一些点子来启发一下你，不过，我相信你也能提出自己的想法。

   - 形容词：crazed, silly, shy, goofy, angry, lazy, obstinate, purple
   - 名词：monkey, elephant, cyclist, teacher, author, hockey player
   - 动词短语：played a ukulele, danced a jig, combed his hair, flapped her ears
   - 副词短语：on the table, at the grocery store, in the shower, after breakfast,with a broom

再给出一个示例输出：“The lazy author combed his hair with a broom”。

2. 编写一个程序，让用户输入名字、年龄、最喜欢的颜色和最喜欢的食物。程序要把所有这 4 项保存在一个文本文件中，每一项分别放在单独的一行上。
3. 完成第 2 题的任务，不过使用 `pickle` 将数据保存到一个文件。（提示：如果先把数据放在一个列表中就会很容易做到。）

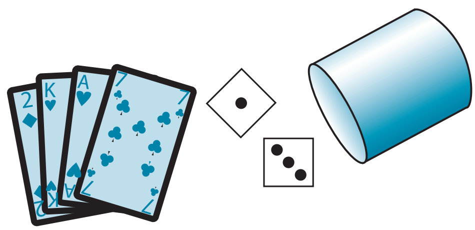
第 23 章

碰运气——随机性

游戏最有意思的一个方面就是你永远也不知道会发生什么。游戏是不可预测的。它们是随机的。正是这种随机性才让游戏很有趣。

我们已经看到，计算机可以模拟随机行为。我们的猜数程序（见第 1 章）使用了 random 模块来生成一个随机整数，也就是要让用户猜的数。另外，你还在第 22 章“动手试一试”中使用了 random 为滑稽句子程序选择单词。

计算机还可以模拟掷骰子或洗牌之类的随机行为。正是因为这一点，我们才有可能创建关于纸牌或骰子（或其他有随机行为的对象）的游戏。例如，几乎所有人都玩过 Windows 上的 Solitaire，这是一个纸牌游戏，每次游戏前程序都会随机地洗牌。另外，Computer Backgammon 游戏也很有名，其中使用了两个骰子。



在这一章中，我们将学习如何使用 random 模块建立计算机生成的骰子和纸牌来玩游戏。这里还会介绍如何使用计算机生成的随机事件来研究概率。所谓概率（probability），就是某件事情发生的可能性。

23.1 什么是随机性

在讨论如何建立有随机行为的程序之前，首先应当了解“随机”到底是什么意思。

以扔硬币为例。如果把一个硬币抛向空中，让它着地，可能正面朝上，也可能背面朝上。一般来说，正面朝上和背面朝上的机会是一样的。有时你会得到正面，有时则是背面。每次抛的时候，你都无法知道会得到什么。因为抛一次的结果不能

预测，我们称之为随机。抛硬币就是随机事件的一个例子。



如果抛硬币的次数很多，可能会发现正面朝上次数和背面朝上次数基本相同。不过这一点永远也不能保证。如果抛 4 次，可能会得到 2 次正面 2 次背面；但是也可能得到 3 次正面 1 次背面，或者 1 次正面 3 次背面，或者甚至连续 4 次正面（或 4 次背面）。如果抛 100 次，可能得到 50 次正面。但是也可能得到 20、44、67 或者甚至 100 次全都是正面！全都是正面的可能性不大，但是确实有可能发生。

这里的关键是每次事件都是随机的。尽管大量抛硬币可能会存在某种规律，但是每一次抛硬币正面朝上或背面朝上的可能性都是一样的。换种说法，也就是说硬币没有记忆。所以即使你刚刚连续抛出了 99 次正面，你可能认为不太可能连续得到 100 个正面，但下一次抛出仍有 50% 的可能会得到正面。这就是随机的含义。

随机事件就是可能有两个或多个结果的事件，你无法预测会得到哪一个结果。这里所说的结果可能是一副牌中的纸牌顺序，或者是掷骰子时的点数，或者是一个硬币哪一面朝上。

23.2 掷骰子

几乎所有人都玩过用到骰子的游戏，可能是 Monopoly、Yahtzee、Trouble、Backgammon 或者别的游戏。不论是哪个游戏，掷骰子都是在游戏中生成随机事件的最常用的方式之一。

骰子在程序中很容易模拟，Python 的 random 模块提供了两种方法来完成这项工作。一种方法是使用 randint() 函数，它会选择一个随机的整数。由于骰子各面上的点数都是整数（1、2、3、4、5 和 6），所以可以这样模拟掷骰子：

```
import random
die_1 = random.randint(1, 6)
```

这会给出介于 1 到 6 之间的一个数，每个数出现的几率相等。这就像是一个真正的骰子。

要完成同样的工作还有一种方法，可以建立所有可能结果的一个列表，然后使用 choice() 函数从这个列表中选择一个结果。具体做法如下：

```
import random
sides = [1, 2, 3, 4, 5, 6]
die_1 = random.choice(sides)
```

这与前一个例子的原理完全相同。choice() 函数随机地从列表中选择一项。在

这里，列表中包含从 1 到 6 的数字。

多个骰子

如果想模拟掷两个骰子呢？如果你只是想把两个骰子的结果相加来得到总数，可能会考虑这样做：

```
two_dice = random.randint(2, 12)
```

毕竟，两个骰子的总和可以是 2 到 12，对不对？嗯，也对也不对。你确实会得到一个介于 2 到 12 之间的随机数，但是不能只是将两个从 1 到 6 的随机数相加来得到。这行代码所做的就像是掷一个有 11 面的大骰子，而不是两个 6 面的骰子。不过这有什么区别呢？这就引入一个主题：概率。要了解二者的差别，最简单的方法就是试一试。

下面我们将掷多次骰子，并跟踪每个面总共出现多少次。这里利用一个循环和一个列表来实现。循环用来掷骰子，列表跟踪每个面出现的次数。下面先来看 11 个面的骰子，如代码清单 23-1 所示。

代码清单 23-1　将一个 11 面的骰子掷 1000 次

```
import random

totals = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]    ← ❶ 列表包含 13 项，索引从 0 到 12
for i in range(1000):
    dice_total = random.randint(2, 12)
    totals[dice_total] += 1    ← ❷ 将这个总数增 1

for i in range (2, 13):
    print "total", i, "came up", totals[i], "times"
```

❶列表的索引是从 0 到 12，不过前两个不会使用，因为我们不关心总数 0 和 1，这是不可能发生的。❷得到一个结果时，我们将相应的列表项增 1。如果结果为 7，就将 `totals[7]` 增 1。所以 `totals[2]` 就是得到 2 的次数，`totals[3]` 是得到 3 的次数。依此类推。

如果运行这个代码，会得到这样的结果：

```
total 2 came up 95 times
total 3 came up 81 times
total 4 came up 85 times
total 5 came up 86 times
total 6 came up 100 times
total 7 came up 85 times
total 8 came up 94 times
total 9 came up 98 times
total 10 came up 93 times
total 11 came up 84 times
total 12 came up 99 times
```

如果查看总数，可以看到所有数出现的次数大致相同，都介于 80 到 100 之间。它们出现的次数并不完全一样，因为这些数都是随机的。不过它们都很接近，而且哪些数出现次数更多并没有明显的规律。你可以运行这个程序，多试几次，确认这一点。或者你也可以把循环次数增加到 10 000 或 100 000 试试看。

现在用两个 6 面的骰子做同样的事情。代码清单 23-2 中的代码会完成这项工作。

代码清单 23-2 将两个 6 面的骰子掷 1000 次

```
import random

totals = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
for i in range(1000):
    die_1 = random.randint(1, 6)
    die_2 = random.randint(1, 6)
    dice_total = die_1 + die_2
    totals[dice_total] += 1

for i in range (2, 13):
    print "total", i, "came up", totals[i], "times"
```

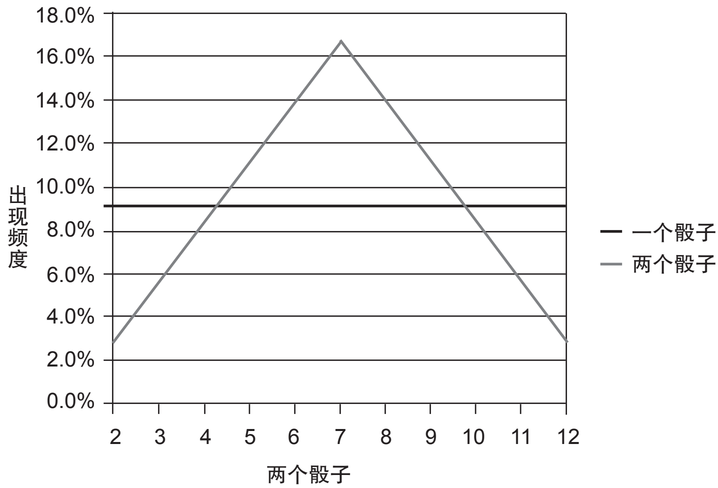
运行这个程序，会得到类似下面的输出：

```
total 2 came up 22 times
total 3 came up 61 times
total 4 came up 93 times
total 5 came up 111 times
total 6 came up 141 times
total 7 came up 163 times
total 8 came up 134 times
total 9 came up 117 times
total 10 came up 74 times
total 11 came up 62 times
total 12 came up 22 times
```

可以注意到，最大的数和最小的数出现比较少，而中间的数字（如 6 和 7）出现得更频繁一些。这与只有一个 11 面骰子的情况有所不同。如果多运行几次，然后计算某个总数出现次数的百分比，会得到右图这样的结果：

结果	一个 11 面的骰子	两个 6 面的骰子
2	9.1%	2.8%
3	9.1%	5.6%
4	9.1%	8.3%
5	9.1%	11.1%
6	9.1%	13.9%
7	9.1%	16.7%
8	9.1%	13.9%
9	9.1%	11.1%
10	9.1%	8.3%
11	9.1%	5.6%
12	9.1%	2.8%

如果画出这些数字的一个图表，可以得到



为什么会有这种差别？原因与概率有关，这是一个庞大的主题。基本说来，对于两个骰子，中间的数更常出现，这是因为掷两个骰子时有更多途径可以得到中间这几个数。

掷两个骰子时，可能发生多种不同组合。以下是这些组合的列表，给出了它们的总数：

1+1 = 2	1+2 = 3	1+3 = 4	1+4 = 5	1+5 = 6	1+6 = 7
2+1 = 3	2+2 = 4	2+3 = 5	2+4 = 6	2+5 = 7	2+6 = 8
3+1 = 4	3+2 = 5	3+3 = 6	3+4 = 7	3+5 = 8	3+6 = 9
4+1 = 5	4+2 = 6	4+3 = 7	4+4 = 8	4+5 = 9	4+6 = 10
5+1 = 6	5+2 = 7	5+3 = 8	5+4 = 9	5+5 = 10	5+6 = 11
6+1 = 7	6+2 = 8	6+3 = 9	6+4 = 10	6+5 = 11	6+6 = 12

共有 36 种可能的组合。现在来看每个总数出现的次数：

- 总数 2 出现 1 次；
- 总数 3 出现 2 次；
- 总数 4 出现 3 次；
- 总数 5 出现 4 次；
- 总数 6 出现 5 次；
- 总数 7 出现 6 次；
- 总数 8 出现 5 次；
- 总数 9 出现 4 次；
- 总数 10 出现 3 次；
- 总数 11 出现 2 次；
- 总数 12 出现 1 次。

这说明，掷出 7 比掷出 2 的途径更多。掷出 1+6、2+5、3+4、4+3、5+2 或 6+1 都可以得到 7。而要得到 2 只有一种情况，也就是掷出 1+1。所以这是有道理的，如果把这两个骰子掷出很多次，得到的 7 会比 2 更多。这也正是我们从两个骰子程序得到的结果。

使用计算机程序生成随机事件是研究概率的一种很好的方法，可以用来完成概率试验，查看需要相当多次尝试才能看到的结果。如果让你把一对真正的骰子掷 1000 次并记录结果，这会花费很长时间，但是计算机程序在远不到 1 秒的时间内就可以办到！

连续 10 次

继续学习接下来的内容之前，下面再做一个概率试验。前面我们讨论过扔硬币，并提到连续多次正面朝上的可能性有多大。为什么不试一试呢？看看连续 10 次正面朝上的情况多久出现一回？这种情况不常发生，所以我们必须扔很多很多次才能看到这种情况出现。为什么不试试扔 1 000 000 次！如果是一个真正的硬币，这可能要花……总之相当长的时间。



如果每 5 秒扔一次硬币，每分钟就会扔 12 次，或者每小时扔 720 次。如果一天扔 12 个小时的硬币（毕竟，你还得睡觉和吃饭），每天可以扔大约 8640 次。所以扔一百万次硬币大约需要 115 天（约 4 个月）。不过利用计算机，我们几秒钟就可以完成。（嗯，也许要几分钟，因为我们还得先写出程序。）

在这个程序中，除了扔硬币，还必须跟踪什么时候可以得到连续 10 次正面朝上。有一种办法是使用一个用来完成统计的变量，即计数器（counter）。

我们需要两个计数器。一个用于统计连续扔出多少个正面朝上，名为 heads_in_row。另一个用于统计连续 10 次正面朝上的情况出现多少次，名为 ten_heads_in_row。程序要做的工作如下：

- 得到正面朝上时，heads_in_row 计数器增 1；
- 正面朝下时，heads_in_row 计数器还原为 0；
- heads_in_row 计数器达到 10 时，将 ten_heads_in_row 计数器增 1，并设置 heads_in_row 计数器还原为 0，重新开始；
- 最后，打印一条消息，指出得到连续 10 次正面朝上的情况出现多少次。

代码清单 23-3 给出了完成这个工作的代码。

代码清单 23-3　查找连续 10 次正面朝上

```
from random import *
coin = ["Heads", "Tails"]
heads_in_row = 0
ten_heads_in_row = 0
for i in range (1000000):
    if choice(coin) == "Heads":        ← 扔硬币
        heads_in_row += 1
    else:
        heads_in_row = 0
    if heads_in_row == 10:
        ten_heads_in_row += 1        ← 连续 10 次正面朝上时，计数器增 1
        heads_in_row = 0

print "We got 10 heads in a row", ten_heads_in_row, "times."
```

运行这个程序时，得到这样的结果：

```
We got 10 heads in a row 510 times.
```

我运行了好几次这个程序，这个数总是在 500 左右。这说明，如果扔 100 万次硬币，连续 10 次正面朝上的情况大约出现 500 次，或者大约每扔 2000 次硬币就会得到连续 10 次正面朝上（1 000 000 / 500 = 2000）。

23.3　创建一副牌

游戏中经常使用的另一种随机事件是抽牌。这是随机的，因为会洗牌，所以你不知道下一张是什么牌。每次洗牌时，顺序都不同。

至于掷骰子和扔硬币，我们说每次扔出都有相同的概率，因为硬币（或骰子）没有记忆。不过纸牌就不同了。从一副牌中抽牌时，剩下的牌越来越少（大多数游戏中都是如此）。这会改变抽出剩余各张牌的概率。

例如，开始时是一整副牌，抽出红桃 4 的机会是 1/52，或者大约 2%。这是因为一副牌里有 52 张牌，而只有一张红桃 4。如果继续抽牌（还没有抽到红桃 4），整副牌只剩下一半时，得到红桃 4 的机会就是 1/26，或者大约 4%。剩下最后一张牌时，如果还没有抽到红桃 4，说明抽出红桃 4 的机会就是 1/1，或者 100%。可以肯定下一个肯定会抽到红桃 4，因为只剩下这一张牌了。

为什么要告诉你所有这些呢？我只是想说明：如果要建立一个利用一副纸牌实现的计算机游戏，就需要在整个过程中跟踪已经从这副牌中取走了哪些牌。要做到这一点有一个很好的方法，就是利用列表。开始时列表中包含一副牌中的所有 52 张

牌，我们使用 random.choice() 函数随机地从这个列表中选牌。每选出一张牌，可以使用 remove() 把它从列表（这副牌）中删除。

洗牌



在一个真正的纸牌游戏中，我们要洗牌，也就是说要把纸牌杂乱地混在一起，让它们有一种随机的顺序。这样一来，我们可以只取最上面的一张牌，这张牌是随机的。不过利用 random.choice() 函数，总能从列表中随机选择。我们不必取“最上面”的牌，所以“洗牌”没有意义。这就像把牌摊开，说“选一张牌，随便哪张都行”。在一个纸牌游戏中，如果每个人都这么做，这会很耗费时间，不过在计算机程序中这非常容易。

纸牌对象

我们要使用一个列表作为“一副牌”。不过这些牌本身怎么表示？如何存储每张牌呢？是存储为字符串还是整数？我们需要知道每张牌的哪些方面？

在纸牌游戏中，我们通常需要知道一张牌的 3 个方面。

- ❒ 花色——方块、红桃、梅花或黑桃。
- ❒ 点数——A、2、3，…10、J、Q、K。
- ❒ 分值——用数字编号的牌（2 到 10），通常分值就等于牌的点数。对于 J、Q 和 K，分值通常是 10，A 的分值可能是 1、11 或者另外某个值，这要依具体游戏而定。

点数	分值	点数	分值
A	1 或 11	8	8
2	2	9	9
3	3	10	10
4	4	J	10
5	5	Q	10
6	6	K	10
7	7		

所以我们要跟踪这 3 个方面，而且需要用某种容器把它们汇集在一起。利用列表可以做到，不过我们还必须记住每一项分别是什么。另一种办法是建立一个包含右面属性的“牌”：

```
card.suit
card.rank
card.value
```

下面就采用这种做法。我们还会增加另外两个属性 suit_id 和 rank_id。

- ❒ suit_id 表示花色，是一个从 1 到 4 的数，其中 1 = 方块，2 = 红桃，3 = 梅花，4 =黑桃。

- rank_id 是从 1 到 13 的数，其中

  1 = A

  2 = 2

  3 = 3

  ⋮

  10 = 10

  11 = J

  12 = Q

  13 = K

增加这两个属性的原因是，这样我们可以很容易地使用一个嵌套 for 循环建立一副 52 张牌。可以用一个内循环对应点数（1 到 13），另外用一个外循环对应花色（1 到 4）。纸牌对象的 __init__() 方法根据 suit_id 和 rank_id 创建其他属性 (花色、点数和分值)。这样还可以很容易地比较两张牌的点数，看哪一张牌的点数更大。

还应当另外增加两个属性，来方便在程序中使用这个纸牌对象。程序需要打印纸牌时，它可能希望打印类似“4H”或“4 of Hearts”（红桃 4）。对于花牌，可能打印成“JD”或“Jack of Diamonds”（方块 J）。我们将增加属性 short_name 和 long_name，这样程序很容易就可以打印纸牌的不同描述（包括短名和长名）。

下面为纸牌建立一个类。见代码清单 23-4。

代码清单 23-4　Card 类

```
class Card:
    def __init__(self, suit_id, rank_id):
        self.rank_id = rank_id
        self.suit_id = suit_id

        if self.rank_id == 1:
            self.rank = "Ace"
            self.value = 1
        elif self.rank_id == 11:
            self.rank = "Jack"
            self.value = 10
        elif self.rank_id == 12:                    创建 rank 和 value 属性
            self.rank = "Queen"
            self.value = 10
        elif self.rank_id == 13:
            self.rank = "King"
            self.value = 10
        elif 2 <= self.rank_id <= 10:
            self.rank = str(self.rank_id)
            self.value = self.rank_id
```

```
        else:
            self.rank = "RankError"
            self.value = -1

        if self.suit_id == 1:
            self.suit = "Diamonds"
        elif self.suit_id == 2:
            self.suit = "Hearts"
        elif self.suit_id == 3:
            self.suit = "Spades"
        elif self.suit_id == 4:
            self.suit = "Clubs"
        else:
            self.suit = "SuitError"
        self.short_name = self.rank[0] + self.suit[0]
        if self.rank == '10':
            self.short_name = self.rank + self.suit[0]
        self.long_name = self.rank + " of " + self.suit
```

创建 suit 属性

❶ 完成一些错误检查

代码❶中的错误检查确保 rank_id 和 suit_id 在正常范围内，而且是整数。否则，在程序中显示纸牌时你就会看到诸如“7 of SuitError”或“RankError of Clubs”之类的错误结果。

设置 short_name 的代码只是取了纸牌点数（6 或者 Jack）的数字或者第一个字母以及花色（Diamonds）的第一个字母，然后将它们放在一起。比如红桃 K（King of Hearts），其 short_name 就是 KH。再比如黑桃 6（6 of Spades），其 short_name 就是 6S。

代码清单 23-4 不是一个完整的程序。这只是 Card 类的类定义。因为这个类可以在不同程序中反复使用，可能应该把它建立为一个模块。把代码清单 23-4 的代码保存为 cards.py。

现在需要建立纸牌的一些实例——实际上，我们完全可以建立一整副牌！要测试我们的 Card 类，下面建立一个程序，创建一副 52 张的牌，然后随机选 5 张并显示它们的属性。代码清单 23-5 提供了相应代码。

代码清单 23-5　建立一副牌

```
import random
from cards import Card

deck = []
for suit_id in range(1, 5):
    for rank_id in range(1, 14):
        deck.append(Card(suit_id, rank_id))
```

导入 Card 模块

❶ 使用嵌套 for 循环建立一副牌

```
hand = []
for cards in range(0, 5):
    a = random.choice(deck)
    hand.append(a)
    deck.remove(a)

print
for card in hand:
    print card.short_name, '=' ,card.long_name, "  Value:", card.value
```

❷ 从一副牌中选 5 张牌作为一手牌

❶内循环处理一种花色中的每张牌，外循环处理每种花色（13 张牌 × 4 种花色 = 52 张牌）。❷然后代码从这副牌中选出 5 张，放在手中（形成一手牌）。另外还要从这副牌中删除选出的这些牌。

如果运行代码清单 23-5 中的代码，应该能得到类似下面的结果：

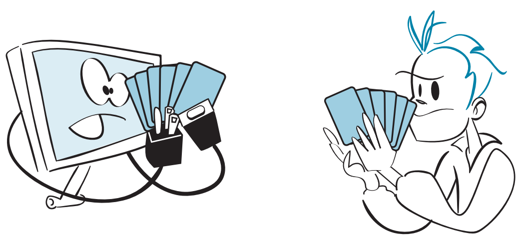
```
7D = 7 of Diamonds    Value: 7
9H = 9 of Hearts    Value: 9
KH = King of Hearts    Value: 10
6S = 6 of Spades    Value: 6
KC = King of Clubs    Value: 10
```

再运行这个代码，会得到 5 张不同的牌。不论你运行多少次，都不会有同一张牌在手中出现两次的情况。

现在我们可以建立一副牌，并且可以从中随机地抽牌，增加到自己手中。听起来已经万事俱备，可以建立一个纸牌游戏了！在下一节，我们会建立一个纸牌游戏，这样你就可以与计算机玩这个游戏了。

23.4　Crazy Eights

你可能听说过一个叫做 Crazy Eights[①]的纸牌游戏，可能还玩过。



计算机上的纸牌游戏都存在一个问题：这些游戏很难有多个玩家。这是因为，在大多数纸牌游戏中，都不希望你看到其他玩家的牌。如果每个人都在看同一台计算机，那么每个人都会看到所有其他人的牌。所以在计算机上玩纸牌游戏时，最好只有两个玩家，也就是你和计算机。Crazy Eights 就是这种适合两个玩家的游戏，下面就来建立一个 Crazy Eights 游戏，用户可以和计算机玩这个游戏。

① 这就类似于“变色龙”纸牌游戏。——译者注

这里给出这个程序的规则。这是一个有两个玩家参与的游戏。每个玩家有 5 张牌，其他的牌都面朝下扣着。翻开一张牌，开始出牌。这个游戏的目标是要在另一个人之前而且在取完一副牌之前出光所有牌。

(1) 每一轮，玩家必须做下面的操作之一：

- 出一张牌，要与翻开的牌花色相同；
- 出一张牌，要与翻开的牌点数相同；
- 出一张 8。

(2) 如果玩家出了一张 8，他可以“叫花色”，这说明他可以选择花色，下一个玩家要根据这个花色出牌。

(3) 如果玩家无法出牌，必须从这副牌中选择一张牌，增加到自己手中。

(4) 如果玩家出光了手中的所有牌，他就赢了，根据另一个玩家手中剩余的牌计算得分：

- 每个 8 得 50 分；
- 每个花牌（J、Q 和 K）得 10 分；
- 每个其他的牌按分值得分；
- 每个 A 得 1 分。

(5) 如果一副牌发光时仍没有人获胜，游戏结束。在这种情况下，每个玩家会根据对方剩余的牌计算得分。

(6) 可以一直玩到达到某个总分，或者直到你累了，得分最高的获胜。

首先要对我们的纸牌对象稍做点修改。Crazy Eights 中的分值与前面基本上一样，只是 8 除外，它的分值是 50 分而不是 8 分。可以修改 `Card` 类中的 `__init__` 方法，让 8 值 50 分，不过这会影响可能用到 `cards` 模块的所有其他游戏。最好在主程序中做这个修改，而类定义不变。我们可以这样做：

```
deck = []
for suit in range(1, 5):
    for rank in range(1, 14):
        new_card = Card(suit, rank)
        if new_card.rank == 8:
            new_card.value = 50
        deck.append(new_card)
```

在这里，将新牌增加到一副牌之前，要检查它是不是一个 8。如果是，就把它的分值设置为 50。

现在已经做好准备，可以具体建立游戏了。程序需要做以下工作。

- 跟踪面朝上的牌。
- 得到玩家的下一步选择（出牌还是抽牌）。

- ❒ 如果玩家想出牌，要确保出牌是合法的：
  - ■ 这张牌必须是合法的牌；
  - ■ 这张牌必须在玩家的手里；
  - ■ 这张牌要与面朝上的牌花色或点数一致，或者是一个 8。
- ❒ 如果玩家出一张 8，叫一个新的花色（并确保选择的是一个合法的花色）。
- ❒ 轮到计算机选择（稍后介绍）。
- ❒ 确定游戏何时结束。
- ❒ 统计得分。

在本章后面，我们会逐条地完成上面的各项工作。其中一些工作只需一行或两行代码就可以完成，有些可能稍长一些。对这些稍长的代码，我们会创建函数，以便从主循环调用。

主循环

介绍具体细节之前，首先要明白程序的主循环。基本说来，玩家和计算机必须轮流选择（出牌或抽牌），直到有人获胜或者双方都无法继续。如代码清单 23-6 所示。

代码清单 23-6 Crazy Eights 的主循环

```
init_cards()
while not game_done:
    blocked = 0
    player_turn()
    if len(p_hand) == 0:
        game_done = True
        print
        print "You won!"
    if not game_done:
        computer_turn()
    if len(c_hand) == 0:
        game_done = True
        print
        print "Computer won!"
    if blocked >= 2:
        game_done = True
        print "Both players blocked.  GAME OVER."
```

轮到玩家（指向 player_turn()）

玩家手中（p_hand）已经没有牌，所以玩家获胜（指向 if len(p_hand) == 0:）

轮到计算机（指向 computer_turn()）

计算机手中（c_hand）已经没有牌，所以计算机获胜（指向 if len(c_hand) == 0:）

❶ 双方都无法继续，所以游戏结束（指向 if blocked >= 2:）

主循环部分要确定游戏何时结束。可能在玩家或计算机出完手上的所有牌时结束，也可能双方手上都还有牌但是都无法继续（也就是说双方都不能合法地出牌），此时游戏也会结束。轮到玩家出牌时，如果玩家无法继续，会在相应代码中设置 `blocked` 变量，轮到计算机出牌时，如果计算机无法继续，同样会在相应代码中设置 `blocked` 变量。❶我们会一直等到 `blocked = 2`，确保玩家和计算机都无法继续。

注意代码清单 23-6 不是一个完整的程序，所以如果试图运行这个代码，你会得

到一条错误消息。这只是一个主循环。我们还需要其他部分来构成一个完整的程序。

这个代码对应一次游戏。如果希望继续玩多次，可以把整个代码包在另一个外部 while 循环中：

```
done = False
p_total = c_total = 0
while not done:
    [play a game... see listing 23.6]
play_again = raw_input("Play again (Y/N)? ")
     if play_again.lower().startswith('y'):
         done = False
     else:
         done = True
```

这就得到了程序的主结构。下面需要增加各个部分来实现我们需要的功能。



像程序员一样思考

前面描述的方法称为“自顶向下”编程方法。

这种方法先从需求大纲开始，然后填入具体细节。

另一种方法叫做“自下而上”编程。采用这种方法时，首先创建各个部分，如“轮到玩家出牌”、“轮到计算机出牌”等，然后把它们放在一起，就像搭积木一样。

这两种方法各有优缺点。究竟选择哪一种方法不是这本书要讨论的主题。但是我想你应当知道可以采用不同的方法来构建一个程序。

明牌

最开始发牌时，要从一副牌中选一张牌翻过来面朝上，作为不要的一堆牌（弃牌堆）中的第一张牌。玩家出牌时，他出的这张牌也要面朝上放在弃牌堆中。弃牌堆中显示的牌叫做明牌（up card）。可以为弃牌堆建立一个列表来跟踪明牌，具体做法与代码清单 23-5 的测试代码中为“一手牌”建立列表相同。不过我们并不关心弃牌堆中的所有牌。我们只关心最后增加的那张牌。所以可以使用 Card 对象的一个实例来跟踪这张牌。

玩家或计算机出牌时，我们会这样做：

```
hand.remove(chosen_card)
up_card = chosen_card
```

当前花色

通常，当前花色就是明牌的花色，玩家或计算机出牌时要与这个花色一致。不过，也有例外。出一张 8 时，玩家可以叫花色。所以如果玩家出了一张方块 8，他可能会叫花色为黑桃。这意味着下一张牌必须是黑桃，尽管现在显示的是方块（方块 8）。

这说明，我们需要跟踪当前花色，因为它可能与现在显示的花色不同。可以使用一个变量 active_suit 来做到：

```
active_suit = card.suit
```

只要出一张牌，我们就会更新当前花色，玩家出一张 8 时，他会选择新的当前花色。

轮到玩家选择

轮到玩家出牌时，首先我们要得到他选择做什么。他可能从手中出一张牌（如果可能的话），或者从这副牌中抽一张牌。如果建立这个程序的一个 GUI 版本，我们会让玩家点击他想出的牌，或者点击这副牌来抽牌。不过现在先建立这个程序的一个基于文本的版本，所以玩家必须键入他的选择，然后我们要检查他键入的内容，明确他想做什么，还要检查输入是否合法。

玩家需要提供什么样的输入呢？为了让你对这些输入有所认识，下面看一个示例游戏。玩家的输入用粗体显示：

```
Crazy Eights

Your hand: 4S, 7D, KC, 10D, QS    Up Card:  6C
What would you like to do?  Type a card name or "Draw" to take a card:  KC
You played the KC (King of Clubs)
Computer plays 8S  (8 of spades) and changes suit to Diamonds

Your hand:  4S, 7D, 10D, QS   Up Card:  8S    Suit:  Diamonds
What would you like to do?  Type a card name or "Draw" to take a card: 10D
You played 10D (10 of Diamonds)
Computer plays QD (Queen of Diamonds)

Your hand:  4S, 7D QS   Up card:  QD
What would you like to do?  Type a card name or "Draw" to take a card: 7D
You played 7D (7 of Diamonds)
Computer plays 9D (9 of Diamonds)

Your hand:  4S, QS   Up card: 9D
What would you like to do?  Type a card name or "Draw" to take a card: QM
That is not a valid card.  Try again:  QD
You do not have that card in your hand.  Try again: QS
That is not a legal play. You must match suit, match rank, play an 8, or draw a card
Try again: Draw
You drew 3C
Computer draws a card

Your hand:  4S, QS, 3C   Up card:  9D
What would you like to do?  Type a card name or "Draw" to take a card: Draw
```

```
You drew 8C
Computer plays 2D

Your hand:  4S, QS, 3C, 8C   Up card:  2D
What would you like to do?  Type a card name or "Draw" to take a card: 8C
You played 8C (8 of Clubs)
Your hand:  4S, QS, 3C   Pick a suit: S
You picked spades
Computer draws a card

Your hand:  4S, QS, 3C   Up card: 8C   Suit:  Spades
What would you like to do?  Type a card name or "Draw" to take a card: QS
You played QS (Queen of Spades)
   .
   .
   .
```

尽管这还不是一个完整的游戏，不过你应该已经有些了解了。玩家必须键入 QS 或 Draw 之类的文本，把他的选择告诉程序。程序要检查玩家键入的内容是合法的。这里将要使用一些字符串方法（第 21 章中介绍的方法）来提供帮助。

显示手中的牌

询问玩家想要做什么之前，我们应当为他显示他手中有哪些牌以及明牌是什么。下面是相关的代码：

```
print "\nYour hand: ",
for card in p_hand:
    print card.short_name,
print "   Up card: ",    up_card.short_name
```

如果出了一张 8，我们还要告诉他当前花色是什么。所以下面再增加几行代码，如代码清单 23-7 所示。

代码清单 23-7　显示玩家手中的牌

```
print "\nYour hand: ",
for card in p_hand:
    print card.short_name,
print "   Up card: ", up_card.short_name
if up_card.rank == '8':
    print"   Suit is", active_suit
```

就像代码清单 23-6 一样，代码清单 23-7 也不是一个完整的程序。我们还需要构建其他部分才能建立一个完整的程序。不过运行代码清单 23-7 中的代码时（作为完整程序的一部分），它会给出类似下面的输出：

```
Your hand:  4S, QS, 3C   Up card: 8C   Suit:  Spades
```

如果想使用纸牌的长名而不是短名，输出会像这样：

```
Your hand:  4 of Spades, Queen of Spades, 3 of Clubs
Up Card:  8 of Clubs    Suit:  Spades
```

在我们的例子中，我们将使用短名。

得到玩家的选择

现在我们需要询问玩家想做什么，并处理他的响应。他主要有两种选择：

- 出一张牌
- 抽一张牌

如果他决定出一张牌，我们需要确保这张牌是合法的。之前说过，需要检查 3 个方面。

- 他选择的是一张合法的牌吗？（他是不是想出一张“蜀葵”4？）
- 这张牌在他手里吗？
- 选择的这张牌能合法出牌吗？（是否与明牌的点数或花色一致，或者是不是一张 8？）

不过如果再考虑一下，可以想到：他手里只能有合法的牌。所以如果我们检查到这张牌确实在他手里，就不用再考虑检查这张牌是否合法。他手里不可能有类似“蜀葵”4 之类的牌，因为这在一副牌中根本不存在。

下面的代码可以得到并验证玩家的选择，见代码清单 23-8。

术语箱

验证（validate）是指确保一样东西是合法的，即允许的或者合理的。

代码清单 23-8　得到玩家的选择

```
print "What would you like to do? ",
response = raw_input ("Type a card to play or 'Draw' to take a card: " )
valid_play = False
while not valid_play:                  <-- 不断尝试直到玩家输入合法的内容
    selected_card = None
    while selected_card == None:       | 得到玩家手中的一
        if response.lower() == 'draw':  | 张牌，或者抽牌
            valid_play = True
            if len(deck) > 0:
                card = random.choice(deck)
                p_hand.append(card)
                deck.remove(card)
                print "You drew", card.short_name
            else:
                print "There are no cards left in the deck"
                blocked += 1
            return
        else:
```

（注释：①如果“抽牌”，从这副牌中取牌，并增加到玩家手中；已经抽牌，所以返回到主循环）

```
            for card in p_hand:
                if response.upper() == card.short_name:  <-- 检查所选的牌是否在玩
                    selected_card = card                      家手中。不断尝试直到
            if selected_card == None:                         确实找到这张牌（否则
                response = raw_input("You don't have that card. Try again:")  要抽牌）

        if selected_card.rank == '8':    <-- 出 8 总是合法的
            valid_play = True
            is_eight = True
        elif selected_card.suit  == active_suit:   <-- 检查选择的牌是否
            valid_play = True                            与明牌花色一致
        elif selected_card.rank  == up_card.rank:  <-- 检查选择的牌是否
            valid_play = True                            与明牌点数一致

        if not valid_play:
            response = raw_input("That's not a legal play.  Try again: ")
```

（这也不是一个完整的、可运行的程序。）

❶在这里，我们会得到一个合法的选择：玩家可能抽牌，也可能出一张合法的牌。如果玩家抽牌，只要这副牌中还有剩余的牌，就在玩家手里增加一张牌。

如果出一张牌，需要从玩家手里删除这张牌，让它成为明牌：

```
p_hand.remove(selected_card)
up_card  = selected_card
active_suit = up_card.suit
print "You played", selected_card.short_name
```

如果出的牌是一张 8，玩家要告诉我们他下一步想要什么花色。因为 player_turn() 函数稍有点长，我们把得到新花色的代码放在一个单独的函数中，名为 get_new_suit()。代码清单 23-9 显示了这个函数的代码。

代码清单 23-9　玩家出一张 8 时得到新花色

```
def get_new_suit():
    global active_suit                  不断尝试直到玩家
    got_suit = False                    输入合法的花色
    while not got_suit:  <--
        suit = raw_input("Pick a suit: ")
        if suit.lower()  == 'd':
            active_suit = "Diamonds"
            got_suit = True
        elif suit.lower()  == 's':
            active_suit = "Spades"
            got_suit = True
        elif suit.lower()  == 'h':
            active_suit = "Hearts"
            got_suit = True
        elif suit.lower()  == 'c':
```

```
            active_suit = "Clubs"
            got_suit = True
        else:
            print"Not a valid suit.  Try again. ",
    print "You picked", active_suit
```

轮到玩家出牌时所要做的就是这些。下一节中，我们要让计算机变得足够聪明来玩这个 Crazy Eights 游戏。

轮到计算机选择

玩家选择之后，就轮到计算机了，所以我们要告诉程序怎么玩 Crazy Eights。它必须与玩家遵循同样的规则，不过程序需要确定出哪一张牌。我们必须专门告诉它如何处理所有可能的情况：

- 出一张 8（并挑选一个新花色）；
- 出另一张牌；
- 抽牌。

为了简化程序，我们要告诉计算机如果有 8 就总是出 8。这可能不是最佳的策略，不过很简单。

如果计算机出了一张 8，它必须挑选新花色。最简单的方法就是统计计算机手中每种花色各有多少张牌，并选择牌数最多的花色。同样，这也不是最完美的策略，不过这样编写代码最为简单。

如果计算机手中没有 8，程序就必须检查所有牌，查看哪些牌可以出。在这些牌中，它会选择出分值最大的牌。

如果根本无法出牌，计算机会抽牌。倘若计算机想要抽牌，但这副牌中已经没有任何牌了，计算机就无法继续，这和人类玩家是一样的。

代码清单 23-10 显示了轮到计算机选择的相应代码，这里给出了一些说明来作出解释。

代码清单 23-10　轮到计算机选择

```
def computer_turn():
    global c_hand, deck, up_card, active_suit, blocked
    options = []
    for card in c_hand:
        if card.rank == '8':              <—— 出一张 8
            c_hand.remove(card)
            up_card  = card
            print "  Computer played ", card.short_name
            #suit totals:  [diamonds, hearts, spades, clubs]
```

```
            suit_totals = [0, 0, 0, 0]
            for suit in range(1, 5):
                for card in c_hand:
                    if card.suit_id == suit:
                        suit_totals[suit-1] += 1
            long_suit = 0
            for i in range (4):
                if suit_totals[i] > long_suit:
                    long_suit = i
            if long_suit == 0:  active_suit = "Diamonds"
            if long_suit == 1:  active_suit = "Hearts"
            if long_suit == 2:  active_suit = "Spades"
            if long_suit == 3:  active_suit = "Clubs"
            print "  Computer changed suit to ", active_suit
            return
        else:
            if card.suit == active_suit:
                options.append(card)
            elif card.rank == up_card.rank:
                options.append(card)

    if len(options) > 0:
        best_play = options[0]
        for card in options:
            if card.value > best_play.value:
                best_play = card

        c_hand.remove(best_play)
        up_card = best_play
        active_suit = up_card.suit
        print "  Computer played ", best_play.short_name


    else:
        if len(deck) >0:
            next_card = random.choice(deck)
            c_hand.append(next_card)
            deck.remove(next_card)
            print "  Computer drew a card"
        else:
            print"  Computer is blocked"
            blocked += 1
    print "Computer has %i cards left" % (len(c_hand))
```

统计每种花色的牌数；牌数最多的花色有个专门的名字叫做“长花色”（long suit）

将长花色作为当前花色

结束计算机的选择，回到主循环

检查可能出哪些牌

检查哪个选择最佳（最高分值）

出牌

抽牌，因为没有任何牌可以出牌

这副牌中已经没有牌了——计算机无法继续

这个程序已经基本上完成了，只需要增加几点就可以了。你可能已经注意到，轮到计算机选择定义为一个函数，而且我们在这个函数中使用了一些全局变量。其实也可以向这个函数传入变量，不过使用全局变量也完全可以，而且与真实世界的实际情况更接近，一副牌是“全局”的——任何人都可以拿到并从中取一张牌。

轮到玩家选择也是一个函数，不过我们还没有显示这个函数定义的第一部分，

这部分是这样的：

```
def player_turn():
    global deck, p_hand, blocked, up_card, active_suit
    valid_play = False
    is_eight = False
    print "\nYour hand: ",
    for card in p_hand:
        print card.short_name,
    print "   Up card: ", up_card.short_name
    if up_card.rank == '8':
        print"   Suit is", active_suit
    print "What would you like to do? ",
    response = raw_input ("Type a card to play or 'Draw' to take a card: " )
```

现在还有一点要做。我们必须跟踪最终谁获胜！

记录分数

要完成这个游戏，还需要最后一点：这就是记录得分。游戏结束时，需要得到赢家的得分，这要根据输家剩余的牌来计算。我们要显示这次游戏的得分，还要显示所有游戏的总分。加入这些内容后，就得到了类似代码清单 23-11 的主循环。

代码清单 23-11　增加了得分的主循环

```
done = False
p_total = c_total = 0
while not done:
    game_done = False
    blocked = 0
    init_cards()                  ← ❶ 建立一副牌，以及玩家和计算机手中的牌
    while not game_done:
        player_turn()
        if len(p_hand) == 0:          ← 玩家获胜
            game_done = True
            print
            print "You won!"
            # display game score here
            p_points = 0
            for card in c_hand:               根据计算机剩余的牌增加得分
                p_points += card.value
            p_total += p_points               ← 将这次游戏的得分增加到总分
            print "You got %i points for computer's hand" % p_points

        if not game_done:
            computer_turn()
        if len(c_hand) == 0:          ← 计算机获胜
            game_done = True
            print
            print "Computer won!"
            # display game score here
```

```
            c_points = 0
            for card in p_hand:                  根据玩家剩余
                c_points += card.value            的牌增加得分
            c_total += c_points                                      将这次游戏
            print "Computer got %i points for your hand" % c_points  的得分增加
        if blocked >= 2:                                             到总分
            game_done = True
            print "Both players blocked.  GAME OVER."
            player_points = 0
            for card in c_hand:
                p_points += card.value
            p_total += p_points                            双方都无法继续，
            c_points = 0                                   所以双方都得分
            for card in p_hand:
                c_points += card.value
            c_total += c_points
            print "You got %i points for computer's hand" % p_points     打印游
            print "Computer got %i points for your hand" % c_points      戏得分
    play_again = raw_input("Play again (Y/N)? ")
    if play_again.lower().startswith('y'):
        done = False
        print "\nSo far, you have %i points" % p_total              打印目前
        print  "and the computer has %i points.\n" % c_total        为止的总分
    else:
        done = True

print "\n Final Score:"                                  打印最
print "You: %i      Computer: %i" % (p_total, c_total)   后总分
```

❶ init_cards() 函数（这里没有显示）的工作只是建立一副牌并创建玩家的一手牌（5 张牌）、计算机的一手牌（5 张牌）以及第一张明牌。

代码清单 23-11 仍然不是一个完整的程序，所以如果你运行这个代码，就会得到一条错误消息。不过如果你一直按我说的做，现在你的编辑器里应该已经有了几乎整个程序。Crazy Eights 的完整代码清单太长了，无法在这里全部列出（大约 200 行代码，还要加上空行和注释），不过你可以在 \Examples 文件夹找到这个代码（如果你使用了本书的安装程序），另外在网站上（www.helloworldbook2.com）也可以找到。可以使用 IDLE 来编辑和运行这个程序。

你学到了什么

在这一章，你学到了以下内容。

- 什么是随机性和随机事件。
- 有关概率的一点内容。
- 如何使用 random 模块在程序中生成随机事件。

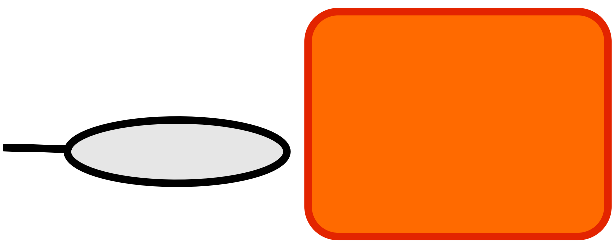
- 如何模拟扔硬币或掷骰子。
- 如何模拟从一副洗过的牌中抽牌。
- 如何玩 Crazy Eights（如果你以前不知道）。

测试题

1. 说明什么是“随机事件”。给出两个例子。
2. 为什么扔一个 11 面（各个面上的数为 2 ～ 12）的骰子与扔两个 6 面的骰子（总和也是 2 ～ 12）不同？
3. 在 Python 中有哪两种方法来模拟掷骰子？
4. 我们使用哪种 Python 变量表示一张牌？
5. 我们使用哪种 Python 变量表示一副牌？
6. 要在抽牌时从一副牌中删除一张牌，或者出牌时从一手牌中删除一张牌，要使用什么方法？

动手试一试

使用代码清单 23-3 的程序试一试“连续 10 次正面朝上”试验，不过可以试试不同的连续次数。多久能出现一次连续 5 个正面朝上？ 6 个呢？ 7 个呢？ 8 个呢？……你发现规律了吗？



第 24 章
计算机仿真

你见过“电子宠物”吧：就是那种小玩具，有一个小小的显示屏，还有一些按钮，宠物饿了可以给它喂吃的，累了让它睡觉，如果它无聊了还能和它玩，诸如此类。你应该见过吧？电子宠物与真实的活的宠物有一些同样的特征。这就是计算机仿真的一个例子——电子宠物设备就是一台微型计算机。

在上一章，我们学习了随机事件以及如何在程序中生成随机事件。从某种角度讲，这就是一种仿真（或模拟）。仿真就是为真实世界的某个东西创建计算机模型。前面已经创建了硬币、骰子和一副牌的计算机模型。

在本章，我们将学习如何使用计算机程序模拟真实世界。

24.1 真实世界建模

为什么要使用计算机对真实世界仿真或建模，这有很多原因。有时出于时间、距离、危险性或其他一些原因，我们要想具体做试验是不实际的。例如，上一章中我们模拟了扔 100 万次硬币。要是把真正的硬币扔这么多次，我们大多数人都没有那么多时间，不过计算机仿真只需几秒钟就能完成。

有时科学家想知道“如果……会怎么样”。如果小行星撞到月球会怎么样？我们不能让一个真正的小行星撞月球，但是计算机仿真可以告诉我们这会有什么后果。月球会不会扩散到太空？会不会撞到地球？会不会改变它的轨道？

飞行员和宇航员学习开飞机和飞船时，他们不能总在真正的飞机和飞船上练习。这样代价太昂贵了！（另外，如果飞行员只是一名“学员”，你真的愿意做他的乘客吗？）所以他们要使用仿真器，仿真器能提供与真正的飞机或飞船同样的控制，让学员进行实践练习。

通过仿真，你可以做很多事情。

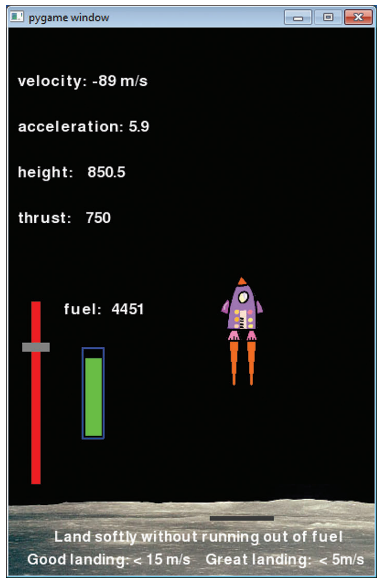
- 你可以做试验或者练习某项技能，而不需要任何设备（除了计算机以外），另外也不会给任何人带来危险。
- 让时间加速或减慢。
- 同时做多个试验。
- 尝试一些可能代价很高、很危险或者在真实世界中不可能实现的事情。

我们打算做的第一个仿真与重力有关。我们想让一个飞船在月球上着陆，不过只有定量的燃料，所以使用推进器必须特别当心。这是一个名叫 Lunar Lander（月球着陆器）的经典游戏的简化版本，Lunar Lander 游戏在多年前相当流行。

24.2 Lunar Lander

开始时飞船离月球表面有一定距离。月球的重力开始把它向下拉，我们必须使用推进器让它的降落放慢，使它平缓着陆。

这个程序看上去是像右图这样的。



左边的小灰条是推进器。用鼠标上下拖动可以控制发动机的推力。燃料表指出你还剩下多少燃料，上面的文本给出了速度、加速度、高度和推力的有关信息。

模拟着陆

为了模拟飞船着陆，必须理解重力和飞船发动机作用力相互之间如何平衡。

在这个仿真中，我们假设重力是恒定的。事实上并不是这样，不过只要飞船离月球不太远，重力几乎是恒定的（对我们的仿真来说非常接近恒定了）。

术语箱

速度（velocity）与速率“speed”含义几乎是一样的，不过速度还包括方向，而速率不包括方向。例如，“每小时 50 公里”描述的是速率，而“每小时向北 50 公里”描述的就是速度。很多人可能会使用“速率”，但实际上他们所指的是“速度”，反之亦然，有些人谈到“速度”时所指的其实是“速率”。在我们的程序中，我们需要知道飞船是向上还是向下，所以会使用速度。

加速度（acceleration）是指速度变化得多快。正的加速度表示速度在增加，负加速度表示速度在减少。

发动机的作用力取决于燃烧了多少燃料。有时这个作用力会大于重力，有时可能比重力小。发动机关闭时，作用力就为 0，此时只剩下重力。

要得到对飞船的总作用力或净作用力，只需把两个作用力相加。因为它们的方向相反，所以一个为正，另一个为负。

一旦得到飞船上的净作用力，可以利用一个公式得出它的速度和位置。

我们的仿真必须跟踪以下几点。

- 飞船距离月球的高度，以及飞船的速度和加速度。
- 飞船的质量（随着燃料的消耗，质量会变化）。
- 发动机的推力或作用力。使用的推力越大，燃料燃烧得就越快。
- 飞船上有多少燃料。推进器燃烧燃料时，飞船会变轻，但是如果所有燃料都耗光，就不再有推力。
- 飞船上的重力。这取决于月球的大小，以及飞船和燃料的质量。

又是 Pygame

我们还是使用 Pygame 建立这个仿真。这里将用 Pygame 的时钟滴答作为我们的时间单位。对于每一个滴答，我们会检查对飞船的净作用力，并更新高度、速度、加速度和剩余的燃料。然后使用这个信息更新图片和文本。

由于动画非常简单，这里不打算用一个动画精灵表示飞船。不过我们会对推进器使用一个精灵（灰色矩形），因为这样就能很容易地用鼠标拖动。燃料表是用 Pygame 的 `draw.rect()` 方法画的两个矩形。文本用 `pygame.font` 对象建立，就像前面 PyPong 中的做法一样。

代码要完成以下工作。

- 初始化游戏——建立 Pygame 窗口，加载图像，为变量设置一些初始值。
- 为推进器定义精灵类。
- 计算高度、速度、加速度和燃料消耗。
- 显示这个信息。
- 更新燃料表。
- 显示火箭尾焰（取决于推力，尾焰大小会改变）。
- 把所有内容“块移”（blit）到屏幕，检查鼠标事件，更新推进器位置，并检查飞船是否已经着陆——这就是主 Pygame 事件循环。
- 显示“游戏结束”和最终统计信息。

代码清单 24-1 显示了 Lunar Lander 的代码，相应的文件是 Listing_24-1.py，可以在 \Examples\LunarLander 文件夹找到这个文件，或者也可以在网站（www.helloworldbook2.com）上找到。在文件夹和网站上还可以找到相关的图片（飞船和月球）。查看代码和说明，一定要确保你能理解所有内容。先不用担心高度、速度和加速度的公式。在高中物理中将会学到这些知识，不过考完试后可能很快就会忘掉（除非你在美国航空航天局工作）。也许这个程序能帮你记住这些公式！

代码清单 24-1 Lunar Lander

```
import pygame, sys

pygame.init()
screen = pygame.display.set_mode([400,600])
screen.fill([0, 0, 0])
ship = pygame.image.load('lunarlander.png')
moon = pygame.image.load('moonsurface.png')
ground  = 540
start = 90
clock = pygame.time.Clock()
ship_mass = 5000.0
fuel = 5000.0
velocity = -100.0
gravity = 10
height = 2000
thrust = 0
delta_v = 0
y_pos = 90
held_down = False
class ThrottleClass(pygame.sprite.Sprite):
    def __init__(self, location = [0,0]):
        pygame.sprite.Sprite.__init__(self)
        image_surface = pygame.surface.Surface([30, 10])
        image_surface.fill([128,128,128])
        self.image = image_surface.convert()
        self.rect = self.image.get_rect()
        self.rect.left, self.rect.centery = location

def calculate_velocity():
    global thrust, fuel, velocity, delta_v, height, y_pos
    delta_t = 1/fps
    thrust = (500 - myThrottle.rect.centery) * 5.0
    fuel -= thrust /(10 * fps)
    if fuel < 0:  fuel = 0.0
    if fuel < 0.1:  thrust = 0.0
    delta_v = delta_t * (-gravity + 200 * thrust / (ship_mass + fuel))
   velocity = velocity + delta_v
    delta_h = velocity * delta_t
    height = height + delta_h
    y_pos = ground - (height * (ground - start) / 2000) - 90
```

注释：
- 降落点是 y = 540（指向 ground = 540）
- 初始化程序（pygame.init() 至 held_down = False）
- 推进器的精灵类（class ThrottleClass 部分）
- “滴答”对应 Pygame 循环的一帧（指向 delta_t = 1/fps）
- 将推进器精灵的 y 位置转换为推力（指向 thrust = ...）
- 根据推力减少燃料（指向 fuel -= ...）
- 物理公式（指向 delta_v = ...）
- 将高度转换为 Pygame y 位置（指向 y_pos = ...）
- 计算高度、速度、加速度和燃料（calculate_velocity 函数）

```
def display_stats():
    v_str = "velocity: %i m/s" % velocity
    h_str = "height:   %.1f" % height
    t_str = "thrust:   %i" % thrust
    a_str = "acceleration: %.1f" % (delta_v * fps)
    f_str = "fuel:  %i" % fuel
    v_font = pygame.font.Font(None, 26)

    v_surf = v_font.render(v_str, 1, (255, 255, 255))
    screen.blit(v_surf, [10, 50])
    a_font = pygame.font.Font(None, 26)
    a_surf = a_font.render(a_str, 1, (255, 255, 255))
    screen.blit(a_surf, [10, 100])
    h_font = pygame.font.Font(None, 26)
    h_surf = h_font.render(h_str, 1, (255, 255, 255))
    screen.blit(h_surf, [10, 150])
    t_font = pygame.font.Font(None, 26)
    t_surf = t_font.render(t_str, 1, (255, 255, 255))
    screen.blit(t_surf, [10, 200])
    f_font = pygame.font.Font(None, 26)
    f_surf = f_font.render(f_str, 1, (255, 255, 255))
    screen.blit(f_surf, [60, 300])
```

使用字体对象显示统计信息

```
def display_flames():
    flame_size = thrust / 15
    for i in range (2):
        startx = 252 - 10 + i * 19
        starty = y_pos + 83
        pygame.draw.polygon(screen, [255, 109, 14], [(startx, starty),
                                (startx + 4, starty + flame_size),
                                (startx + 8, starty)], 0)
```

画出尾焰三角形

使用两个三角形显示火箭尾焰

```
def display_final():
    final1 = "Game over"
    final2 = "You landed at %.1f m/s" % velocity
    if velocity > -5:
        final3 = "Nice landing!"
        final4 = "I hear NASA is hiring!"
    elif velocity > -15:
        final3 = "Ouch!  A bit rough, but you survived."
        final4 = "You'll do better next time."
    else:
        final3 = "Yikes!  You crashed a 30 Billion dollar ship."
        final4 = "How are you getting home?"
    pygame.draw.rect(screen, [0, 0, 0], [5, 5, 350, 280],0)
    f1_font = pygame.font.Font(None, 70)
    f1_surf = f1_font.render(final1, 1, (255, 255, 255))
    screen.blit(f1_surf, [20, 50])
    f2_font = pygame.font.Font(None, 40)
    f2_surf = f2_font.render(final2, 1, (255, 255, 255))
    screen.blit(f2_surf, [20, 110])
    f3_font = pygame.font.Font(None, 26)
    f3_surf = f3_font.render(final3, 1, (255, 255, 255))
    screen.blit(f3_surf, [20, 150])
```

游戏结束时显示最终统计信息

```
    f4_font = pygame.font.Font(None, 26)
    f4_surf = f4_font.render(final4, 1, (255, 255, 255))
    screen.blit(f4_surf, [20, 180])
    pygame.display.flip()

myThrottle = ThrottleClass([15, 500])
running = True
while running:
    clock.tick(30)
    fps = clock.get_fps()
    if fps < 1:  fps = 30
    if height > 0.01:
        calculate_velocity()
        screen.fill([0, 0, 0])
        display_stats()
        pygame.draw.rect(screen, [0, 0, 255], [80, 350, 24, 100], 2)
        fuelbar = 96 * fuel / 5000
        pygame.draw.rect(screen, [0,255,0],
                [84,448-fuelbar,18, fuelbar], 0)
        pygame.draw.rect(screen, [255, 0, 0],
                [25, 300, 10, 200],0)
        screen.blit(moon, [0, 500, 400, 100])
        pygame.draw.rect(screen, [60, 60, 60],
                [220, 535, 70, 5],0)
        screen.blit(myThrottle.image, myThrottle.rect)
        display_flames()
        screen.blit(ship, [230, y_pos, 50, 90])
        instruct1 = "Land softly without running out of fuel"
        instruct2 = "Good landing: < 15m/s   Great landing: < 5m/s"
        inst1_font = pygame.font.Font(None, 24)
        inst1_surf = inst1_font.render(instruct1, 1, (255, 255, 255))
        screen.blit(inst1_surf, [50, 550])
        inst2_font = pygame.font.Font(None, 24)
        inst2_surf = inst1_font.render(instruct2, 1, (255, 255, 255))
        screen.blit(inst2_surf, [20, 575])
        pygame.display.flip()

    else:
        display_final()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEBUTTONDOWN:
            held_down = True
        elif event.type == pygame.MOUSEBUTTONUP:
            held_down = False
        elif event.type == pygame.MOUSEMOTION:
            if held_down:
                myThrottle.rect.centery = event.pos[1]
                if myThrottle.rect.centery < 300:
                    myThrottle.rect.centery = 300
                if myThrottle.rect.centery > 500:
                    myThrottle.rect.centery = 500
pygame.quit()
```

创建推进器对象

主 Pygame 事件循环开始

画出燃料表轮廓

燃油量

画出推进器滑块

画出月球

降落点

画出推力操纵杆

画出飞船

画出所有内容

游戏结束。打印最终得分

检查鼠标是否拖动推进器

更新推进器位置

试着运行这个程序。没准你会发现自己是一个不错的飞船驾驶员！如果你认为这太简单了，可以修改代码，让重力更大一些，使飞船更重（质量更大），或者减少一些燃料，还可以设置一个不同的起始高度或速度。你是程序员，所以游戏该怎么做由你来决定。

Lunar Lander 仿真主要考虑重力。在本章后面的内容，我们将讨论仿真中另一个重要的因素——时间。我们会建立一个需要跟踪时间的仿真。

24.3 跟踪时间

在很多仿真中，时间是一个重要的因素。有时我们希望时间加快，或者让事情比真实世界中发生得更快，这样就不必等待那么长时间才能得出会发生什么。有时可能希望慢下来，因为有些事情通常发生得太快让人来不及观察，通过让时间减慢，就能更好地观察这样一些事情。有些时候则希望程序保持实时（real time）——就是与真实世界中保持一致。不论哪种情况，我们都需要用某种时钟在程序中度量时间。

每个计算机都内置有一个时钟，可以用来度量时间。前面我们已经见过几个使用和度量时间的例子。

- 在第 8 章，我们使用 `time.sleep()` 函数建立了一个倒计时的定时器。
- 在我们完成的几个 Pygame 程序中，使用了 Pygame 的 `time.delay` 和 `clock.tick` 函数来控制动画速度或帧速率。还使用 `get_fps()` 检查动画运行的快慢，这也是一种度量时间的方法（每一帧的平均时间）。

到目前为止，我们总是在程序运行时跟踪时间，不过有时还需要在程序不运行时跟踪时间。如果在 Python 中建立一个电子宠物（Virtual Pet）程序，你可能并不希望让它一直都在运行。你会玩一会，然后停止程序，以后再玩。在你离开期间，宠物可能会累或者会饿，或者会去睡觉。所以程序需要知道从最后一次运行以来已经过去了多长时间。

要做到这一点，可以让程序在关闭之前将信息（当前时间）保存到文件中。这样一来，下一次启动时，程序可以读取这个文件，得到原来的时间，并检查当前时间，比较这两个时间从而得出从程序上一次运行以来已经过去了多长时间。

Python 提供了一种特殊的对象来处理时间和日期。我们将在下一节更详细地学习 Python 的日期和时间对象。

术语箱

> 将当前时间保存到文件中以备以后读取，这称为一个时间戳（timestamp）。

24.4 时间对象

Python 的日期和时间对象类在单独的 datetime 模块中定义。datetime 模块包含处理日期、时间以及日期或时间之差（delta）的类。

术语箱

> delta 的含义是“差”。这是一个希腊字母，看起来像是一个三角形（Δ）。
>
> 科学和数学领域经常使用希腊字母作为某些量的简写。delta 用于表示两个值之差。

我们要使用的第一种对象是 datetime 对象。（没错，这个类与模块同名。）datetime 对象包含年、月、日、小时、分和秒。可以像这样创建一个 datetime 对象（在交互模式中）：

```
>>> import datetime
>>> when = datetime.datetime(2012, 10, 24, 10, 45, 56)
```

（图注：datetime.datetime 中第一个 datetime 为“模块名”，第二个 datetime 为“类名”）

下面来看会得到什么：

```
>>> print when
2012-10-24 10:45:56
```

我们创建了一个 datetime 对象，名为 when，其中包含日期和时间值。

创建一个 datetime 对象时，参数的顺序（括号中的数）应当是年、月、日、小时、分和秒。不过如果你记不住这个顺序，也可以按任意顺序放置参数，只是要告诉 Python 各个参数分别表示什么，如下：

```
>>> when = datetime.datetime(hour=10, year=2012, minute=45, month=10,
                             second=56, day=24)
```

还可以对 datetime 对象做一些其他处理，你可以得到单个部分，比如年、日或者分。还可以得到日期和时间的一个格式化字符串。在交互模式中试试下面的代码：

```
>>> print when.year
2012
>>> print when.day
24
>>> print when.ctime()
Wed Oct 24 10:45:56 2012
```

（图注：前两条语句——“得到 datetime 对象的单个部分”；when.ctime() 的输出——“打印字符串版本的日期和时间”）

datetime 对象分日期类和时间类。如果只关心日期，可以使用 date 类，其中只有年、月和日。如果只关心时间，可以使用 time 类，其中只包括小时、分和秒。

如下所示：

```
>>> today = datetime.date(2012, 10, 24)
>>> some_time = datetime.time(10, 45, 56)
>>> print today
2012-10-24
>>> print some_time
10:45:56
```

类似于 datetime 对象，如果指定了各个参数分别表示什么，完全可以按不同的顺序传入参数：

```
>>> today = datetime.date(month=10, day=24, year=2012)
>>> some_time = datetime.time(second=56, hour=10, minute=45)
```

还有一种方法可以把 datetime 对象分解为 date 对象和 time 对象：

```
>>> today = when.date()
>>> some_time = when.time()
```

另外可以使用 datetime 模块中 datetime 类的 combine() 方法把 date 和 time 对象结合起来构成 datetime 对象：

```
>>> when = datetime.datetime.combine(today, some_time)
```

模块名（datetime） 类名（datetime） 方法（combine）

我们已经知道了什么是 datetime 对象，也了解了它的一些属性，下面来看如何比较两个 datetime 对象，得到它们的差（两个时间之间间隔多长）。

两个时间之差

在仿真中，我们常常需要知道经过了多长时间。例如，在一个电子宠物程序中，可能需要知道上一次给宠物喂食之后过去了多长时间，来确定它是不是饿了。

datetime 模块为此提供了一个对象类，可以帮助我们得出两个日期或时间之差。这个类名为 timedelta。应该记得 delta 表示“差”。所以 timedelta 就是两个时间之差。

要创建一个 timedelta，得到两个时间之差，只需要将这两个时间相减，如下：

```
>>> import datetime
>>> yesterday = datetime.datetime(2012, 10, 23)
>>> tomorrow = datetime.datetime(2012, 10, 25)
>>> difference = tomorrow - yesterday      <- 得到两个日期之差
>>> print difference
2 days, 0:00:00                            <- 明天和昨天相差 2 天
>>> print type(difference)
<type 'datetime.timedelta'>                <- 这个差是一个 timedelta 对象
```

注意，将两个 datetime 对象相减时，我们得到的不是另一个 datetime，而是一个 timedelta 对象。Python 会自动完成这一点。

小段时间

到目前为止，我们一直都在讨论按整秒度量的时间。但是时间对象（date、time、datetime 和 timedelta）比这更精确。它们可以精确度量到微秒级，也就是百万分之一秒。

要了解这一点，可以试试 now() 方法，它会给出计算机时钟的当前时间：

```
>>> print datetime.datetime.now()
2012-10-24 21:25:44.343000
```

注意这个时间不仅仅包含秒，还包括不到 1 秒的部分：

```
44.343000
```

我的计算机上，最后 3 位总是 0，因为我的操作系统的时钟只能精确到毫秒（千分之一秒）。不过对我来说这已经足够精确了！

有一点很重要，尽管秒部分看起来像是浮点数，但它实际上存储为秒数（整数）和微秒数（整数），也就是 44 秒和 343 000 微秒。要把它转换为浮点数还需要一个小公式。假设有一个名为 some_time 的时间对象，如果希望按浮点数形式得到秒数，相应的公式如下：

```
seconds_float = some_time.seconds + some_time.microseconds / float(1000000)
```

这里使用 float() 函数来确保不会遭遇整数相除问题。



可以使用 now() 方法和一个 timedelta 对象来测试你的打字速度。代码清单 24-2 中的程序会显示一条随机消息，用户必须键入这条消息。程序将检查用户键入这条消息所用的时间，然后计算出打字速度。你可以试试看。

代码清单 24-2 度量时间差——打字速度测试

```
import time, datetime, random      ← 为使用 sleep() 函数，
                                     导入 time 模块
messages = [
    "Of all the trees we could've hit, we had to get one that hits back.",
    "If he doesn't stop trying to save your life he's going to kill you.",
    "It is our choices that show what we truly are, far more than our abilities.",
```

```
    "I am a wizard, not a baboon brandishing a stick.",
    "Greatness inspires envy, envy engenders spite, spite spawns lies.",
    "In dreams, we enter a world that's entirely our own.",
    "It is my belief that the truth is generally preferable to lies.",
    "Dawn seemed to follow midnight with indecent haste."
    ]

print "Typing speed test. Type the following message. I will time you."
time.sleep(2)
print "\nReady..."
time.sleep(1)
print "\nSet..."
time.sleep(1)
print "\nGo:"
message = random.choice(messages)
print "\n " + message
start_time = datetime.datetime.now()
typing = raw_input('>')
end_time = datetime.datetime.now()
diff = end_time - start_time
typing_time = diff.seconds + diff.microseconds / float(1000000)
cps = len(message) / typing_time
wpm = cps * 60 / 5.0
print "\nYou typed %i characters in %.1f seconds." % (len(message),
                              typing_time)
print "That's %.2f chars per sec, or %.1f words per minute" %(cps, wpm)
if typing == message:
    print "You didn't make any mistakes."
else:
    print "But, you made at least one mistake."
```

为使用 Sleep() 函数，导入 time 模块

打印指令

从列表中选取消息

启动时钟

停止时钟

计算经过的时间

计算打字速度时，1 个词 = 5 个字符

利用打印格式化显示结果

关于 timedelta 对象还有一点应当知道。与 datetime 对象不同（datetime 对象包含年、月、日、小时、分和秒（以及微秒）），timedelta 对象只有日、秒和微秒。如果想得到月或年，必须根据天数计算出来。如果希望得到分或小时数，必须根据秒数来计算。

24.5 把时间保存到文件

在本章最前面我们提到过，有时需要把一个时间值保存到（硬盘上的）文件中，这样一来，即使程序没有运行，这条信息也能得到保存。如果程序结束时保存当前时间（now()），程序再次启动时就可以检查这个时间，并打印这样的一条消息：

```
It has been 2 days, 7 hours, 23 minutes since you last used this program.
```

当然，大多数程序不会这样做，不过确实有一些程序需要知道已经有多长时间空闲（没有运行），电子宠物程序就是这样一个例子。就像几年前流行的电子宠物钥匙链一样，你可能希望即使你没有使用程序，它仍然会跟踪时间。例如，如果你结束程序之后过了两天再来看你的电子宠物，它应该会非常饿！程序要知道宠物有多饿，只有一个办法，就是要知道从最后一次喂食到现在隔了多长时间。这也包括程序关闭的时间。

将时间保存到一个文件中有两种方法。可以把一个字符串直接写入文件，如下：

```
timeFile.write ("2012-10-24 14:23:37")
```

要读这个时间戳时，可以使用一些字符串方法（如 split()）将这个字符串分解为各个部分，如天、月、年以及小时、分和秒。这种做法应该是可行的。

另一种方法是使用 pickle 模块，这在第 22 章介绍过。pickle 模块允许你把任何类型的变量保存到文件中，也包括对象。由于我们要使用 datetime 对象跟踪时间，所以使用 pickle 可以很容易地把时间对象存入文件，还能很方便地读取。

下面来看一个非常简单的例子，它会打印一条消息，指出程序最后一次运行的时间。这个程序要完成下面的工作。

- 查找一个 pickle 文件并打开这个文件。Python 有一个 os（操作系统 operating system 的简写）模块，可以告诉我们这个文件是否存在。这里要使用的方法名为 isfile()。
- 如果文件存在，就认为程序之前运行过，得出它最后一次运行的时间（根据 pickle 文件中的时间得出）。
- 然后用当前时间写一个新的 pickle 文件。
- 如果这是程序第一次运行，就没有 pickle 文件可以打开，所以会显示一条消息，指出我们创建了一个新的 pickle 文件。

代码清单 24-3 给出了这个程序的代码。可以试试看结果如何。

代码清单 24-3　使用 pickle 把时间保存到文件中

```
import datetime, pickle            导入 datetime、
import os                          pickle 和 os 模块

first_time = True
if os.path.isfile("last_run.pkl"):        ← 检查 pickle 文件是否存在
    pickle_file = open("last_run.pkl", 'r')     ← 打开 pickle 文件行进行读取（如果文件存在）
    last_time = pickle.load(pickle_file)        ← 还原 datetime 对象
    pickle_file.close()
    print "The last time this program was run was ", last_time
    first_time = False

pickle_file = open("last_run.pkl", 'w')     ← 打开（或创建）pickle 文件来写入信息
pickle.dump(datetime.datetime.now(), pickle_file)     ← 存入当前时间的 datetime 对象
pickle_file.close()
if first_time:
    print "Created new pickle file."
```

现在已经万事俱备，可以建立简单的电子宠物程序了，下一节就来建立这样一个程序。

24.6 电子宠物

我们将要建立一个简化了的电子宠物程序，正如前面所说的一样，这是一种仿真。你可以购买电子宠物玩具（比如有一个小屏幕的钥匙链），下载电子宠物软件，还有一些网站（如 Neopets 和 Webkinz），就采用了电子宠物的形式。当然，所有这些也都是仿真。它们会模仿一些真实动物的行为，会饿，会感到孤单，会觉得累。要让它们快乐健康，你必须给它们喂食，和它们玩，还要带它们看病。

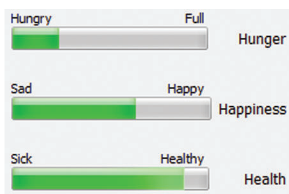
我们的电子宠物会简单得多，与你购买或在线玩的电子宠物相比没有那么真实，因为我只是想让你有一些基本认识，而且我不希望代码太过复杂。不过你可以在这个简化版本的基础上，根据你的想法进行扩展或改进。

我们的程序要具备以下特性。

- ❒ 对这个宠物可以有 4 种活动：给它喂食、带它散步、和它玩或者带它看病。



- ❒ 可以监测这个宠物的 3 种统计信息：饥饿感、快乐度和健康度。



- ❒ 宠物可以醒着或者睡觉。



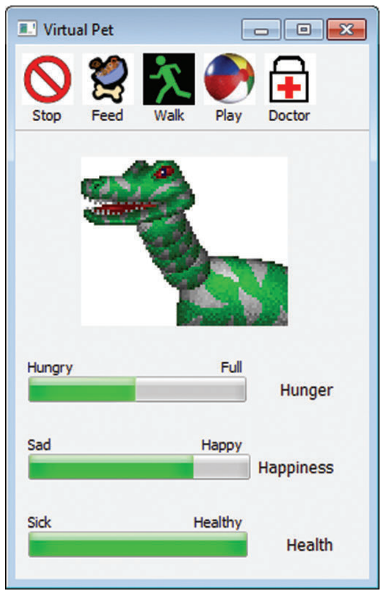

- ❒ 饥饿感会随时间增加。可以通过喂食减少饥饿感。
- ❒ 宠物睡觉时饥饿感的增加会减慢。
- ❒ 如果宠物在睡觉，你做任何活动都会让它醒过来。
- ❒ 如果宠物太饿了，它的快乐度会减少。
- ❒ 如果宠物实在太饿了，它的健康度会减少。
- ❒ 带宠物散步会同时增加它的快乐度和健康度。

- 与宠物玩会让它的快乐度增加。
- 带宠物看病会让它的健康度增加。
- 宠物有 6 个不同的图片：
    - 一个睡觉的图片；
    - 一个醒着但什么也不做的图片；
    - 一个散步的图片；
    - 一个玩耍的图片；
    - 一个进食的图片；
    - 一个看病的图片。

图片可以使用一些简单的动画。后面几节我们将看到如何把所有这些整合在一起构成一个程序。

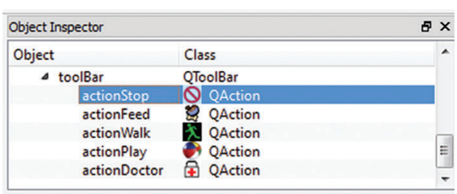
GUI

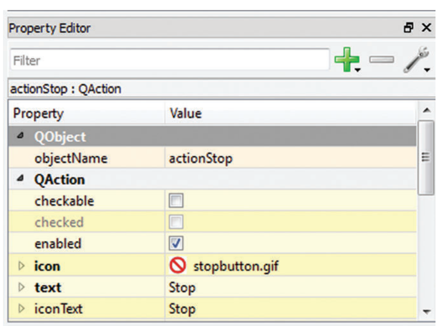
卡特和我为我们的电子宠物程序创建了一个 PyQt GUI。其中有一些按钮（实际上是工具条上的图标）用来完成活动，还有一些进度条显示重要的统计信息。另外还留有一个位置显示宠物的图片（宠物正在做什么）。

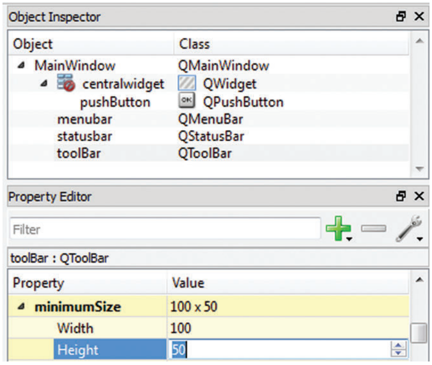


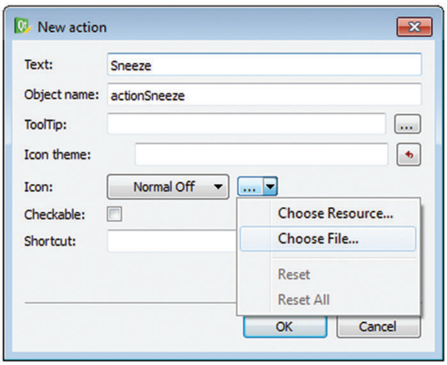
注意，窗口的标题栏上写着 Virtual Pet（虚拟宠物）。怎么设置窗口标题呢？在 Qt Designer 应用中新建一个窗口，然后在对象检查器中点击 MainWindow 对象。之后，在属性编辑器中找到 windowTitle 属性，将它改为 Virtual Pet（或者你想在标题栏中显示的任意文字）。

用来控制宠物活动的一组按钮是 PyQt 中一种叫做工具栏（Toolbar）的组件。工具栏和菜单栏一样也有行为，但不同之处在于，工具栏中的每个行为都有一个与之关联的图标。









要添加一个工具栏，右键点击主窗口，然后选择 Add Toolbar（添加工具栏）。这会在窗口顶部创建一个非常小的工具栏。在对象检查器中找到工具栏，点击它。然后在属性编辑器中找到 minimumSize 属性，将它的宽设为 100，高设为 50。

要将行为（图标）添加到工具栏上，点击 Qt Designer 右下角的 Action Editor（行为编辑器）标签。在 Action Editor 面板中任意位置点击右键，然后选择 New（新建）。你会看到一个用来添加新行为的对话框。你唯一需要输入的是 Text，Qt Designer 会自动填写对象名称。然后在中间找到三个点（...）的小按钮，点击右边的向下箭头，选择 Choose File，接着选择你想在工具栏按钮上使用的图片文件。

要在工具栏中添加新图标还有最后一步要做。一旦你创建了新的行为，就能在 Action Editor 的列表中看到它。现在你需要将它拖到工具栏上。当你拖过来时，你为新行为选择的图像将会作为工具栏的新图标出现。Qt Designer 会自动缩放图片以适应工具栏的大小。

血量是名为 Progress Bar（进度条）的组件类型。主图像是一个 Push Button（我们之前用过），通过设置它的属性，可以让它看起来不像普通的按钮，而是显示一幅图像。

窗口中其余部分的文本是 Label 组件。

你可以像这样使用 PyQt Designer 创建一个 GUI。你也可以（从示例文件夹中）将我们创建好的 GUI 加载到 Qt Desinger 中，以检查这些组件和它们的属性。

算法

要为电子宠物程序写代码，需要更明确地了解宠物的行为。以下是我们要使用的算法。

- 我们把宠物的一“天”分为 60 个部分，每一部分称为一个“滴答”。每个滴答的实际时间是 5 秒钟，所以宠物的“一天”就是我们实际时间的 5 分钟。

- 宠物在 48 个滴答中都醒着，然后它想睡 12 个滴答。你可以把它叫醒，不过这样会让它很不高兴！
- 饥饿感、快乐度和健康度的范围都是 0 到 8。
- 醒着时，饥饿感每个滴答会增加 1 个单位，快乐度每 2 个滴答减少 1 个单位（除非在散步或者玩）。
- 睡觉时，饥饿感每 3 个滴答增加 1 个单位。
- 进食时，饥饿感每个滴答减少 2 个单位。
- 玩时，快乐度每个滴答增加 1 个单位。
- 散步时，快乐度和健康度每 2 个滴答增加 1 个单位。
- 看病时，健康度每个滴答增加 1 个单位。
- 如果饥饿感达到 7，健康度每 2 个滴答减少 1 个单位。
- 如果饥饿感达到 8，健康度每个滴答减少 1 个单位。
- 如果睡觉时被叫醒，快乐度减少 4 个单位。
- 如果程序不在运行，宠物可能醒着（什么也不做），也可能在睡觉。
- 程序重启时，我们会统计过去了多少滴答，并对应过去的每个滴答更新统计信息。

看起来好像规则很多，不过编写代码其实很容易。实际上，你可能还想增加更多的行为，让它更加有趣。稍后就会给出代码（还会做一些解释）。

简单动画

并不总是需要 Pygame 才能完成动画。我们可以在 PyQt 中通过使用定时器完成简单的动画。定时器每隔一段时间会创建一个事件。可以编写一个事件处理器，在定时器到时间时让某个事情发生。这就类似于为一个用户动作编写事件处理器，比如说点击一个按钮，只不过定时器事件是由程序（而不是用户）生成的。定时器到时间时生成的事件类型是 `timeout` 事件。

我们的电子宠物 GUI 将使用两个定时器：一个用于动画，另一个用于滴答。动画每半秒（0.5 秒）更新一次，滴答每 5 秒发生一次。

动画定时器时间到时，我们会所显示宠物的图像。每个活动（进食、玩等）都有自己的一组图像来实现动画，每组图像将存储在一个列表中。动画会循环显示这个列表中的所有图像。程序将根据正在进行的活动来确定使用哪个列表。

试一试，再试一试

这个程序中还要使用一个新内容，这称为 `try-except` 块。

如果程序要做一件事情，而且这个事情有可能导致错误，那么最好提供一种办

法来收集错误消息并进行处理，而不是让程序直接停止。这可以利用 try-except 块来做到。

例如，如果想打开一个文件，但是这个文件并不存在，你就会得到一条错误消息。如果你没有处理这个错误，程序会在这里停止。不过，也许你想让用户重新输入文件名（没准她只是敲错了）。利用 try-except 块，你可以获取到错误信息并继续执行。

对于打开文件的例子，try-except 块如下所示：

```
try:
    file = open("somefile.txt", "r")
except:
    print "Couldn't open the file.  Do you want to reenter the filename?"
```

你想尝试的部分（可能导致一个错误）要放在 try 块中。在这个例子中就是尝试打开一个文件。如果可以打开文件而不会导致错误，就会跳过 except 部分。

如果 try 块中的代码确实导致一个错误，就会运行 except 块中的代码。except 块中的代码告诉程序一旦出现错误该做些什么。你可以这样来考虑：

```
try:
    做这件事 （不做其他事情……）
except:
    如果有错误，就做这件事
```

try-except 语句是 Python 处理错误所采用的方法，这通常称为错误处理（error handling）。错误处理允许你编写可能出错的代码（甚至是很严重的错误，倘若没有错误处理，这些错误在正常情况下甚至会让你的程序停止），使程序仍能继续运行。我们不打算在这本书里更详细地讨论错误处理，不过我希望你能了解一些基础知识，因为在电子宠物代码中就会看到错误处理。

下面来看这个代码，见代码清单 24-4。这里的说明已经对大部分工作做了解释。这个代码有点长，所以如果你不想自己键入，可以在 \Examples\VirtualPet 文件夹找到这个程序（如果你运行了本书的安装程序）。也可以从这本书的网站（www.helloworldbook2.com）下载。PyQt UI 文件和所有图片也都已经提供。试着运行这个程序，然后再看代码，确保你能理解它是如何工作的。

代码清单 24-4　VirtualPet.py

```
import sys, pickle,datetime
from PyQt4 import QtCore, QtGui, uic

formclass, baseclass = uic.loadUiType("mainwindow.ui")

class MyForm(baseclass, formclass):
    def __init__(self, parent=None):
        QtGui.QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.doctor = False
        self.walking = False
        self.sleeping = False
        self.playing = False
        self.eating = False                    初始化值
        self.time_cycle = 0
        self.hunger = 0
        self.happiness  = 8
        self.health = 8
        self.forceAwake = False
        self.sleepImages = ["sleep1.gif","sleep2.gif","sleep3.gif",
                            "sleep4.gif"]
        self.eatImages = ["eat1.gif", "eat2.gif"]
        self.walkImages = ["walk1.gif", "walk2.gif", "walk3.gif",
                           "walk4.gif"]
        self.playImages = ["play1.gif", "play2.gif"]
        self.doctorImages = ["doc1.gif", "doc2.gif"]
        self.nothingImages  = ["pet1.gif", "pet2.gif", "pet3.gif"]
                                                用于动画的列表图像

        self.imageList = self.nothingImages
        self.imageIndex = 0

        self.actionStop.triggered.connect(self.stop_Click)
        self.actionFeed.triggered.connect(self.feed_Click)
        self.actionWalk.triggered.connect(self.walk_Click)
        self.actionPlay.triggered.connect(self.play_Click)
        self.actionDoctor.triggered.connect(self.doctor_Click)
                                                将事件处理器连接到工具条按钮

        self.myTimer1 = QtCore.QTimer(self)
        self.myTimer1.start(500)
        self.myTimer1.timeout.connect(self.animation_timer)
                                                设置定时器
        self.myTimer2 = QtCore.QTimer(self)
        self.myTimer2.start(5000)
        self.myTimer2.timeout.connect(self.tick_timer)

        filehandle = True
        try:
            file = open("savedata_vp.pkl", "r")       尝试打开pickle 文件
        except:
            filehandle = False
        if filehandle:
            save_list = pickle.load(file)      <---  如果文件打开，从pickle 文件读取
            file.close()
```

```
            else:
                save_list = [8, 8, 0, datetime.datetime.now(), 0]
            self.happiness = save_list[0]
            self.health    = save_list[1]
            self.hunger    = save_list[2]
            timestamp_then = save_list[3]
            self.time_cycle = save_list[4]

            difference = datetime.datetime.now() - timestamp_then
            ticks = difference.seconds / 50
            for i in range(0, ticks):
                self.time_cycle += 1
                if self.time_cycle == 60:
                    self.time_cycle = 0
                if self.time_cycle <= 48:
                    self.sleeping = False
                    if self.hunger < 8:
                        self.hunger += 1
                else:
                    self.sleeping = True
                    if self.hunger < 8 and self.time_cycle % 3 == 0:
                        self.hunger += 1
                if self.hunger == 7 and (self.time_cycle % 2 ==0) \
                               and self.health > 0:
                    self.health -= 1
                if self.hunger == 8 and self.health > 0:
                    self.health -=1
            if self.sleeping:
                self.imageList = self.sleepImages
            else:
                self.imageList = self.nothingImages

        def sleep_test(self):
            if self.sleeping:
                result = (QtGui.QMessageBox.warning(self, 'WARNING',
    "Are you sure you want to wake your pet up?  He'll be unhappy about it!",
                      QtGui.QMessageBox.Yes | QtGui.QMessageBox.No,
                      QtGui.QMessageBox.No))

                if result == QtGui.QMessageBox.Yes:
                    self.sleeping = False
                    self.happiness -= 4
                    self.forceAwake = True
                    return True
                else:
                    return False
            else:
                return True

        def doctor_Click(self):
            if self.sleep_test():
                self.imageList = self.doctorImages
                self.doctor = True
                self.walking = False
                self.eating = False
                self.playing = False
```

如果 pickle 文件没有打开，使用默认值

从列表取出单个的值

检查从最后一次运行以来过去了多长时间

醒着

睡觉

模拟关机期间发生的所有滴答

使用正确的动画——醒着或者在睡觉

对话类型

要显示的按钮

默认按钮

做动作之前检查宠物是否在睡觉

医生按钮事件处理器

```
    def feed_Click(self):
        if self.sleep_test():
            self.imageList = self.eatImages
            self.eating = True
            self.walking = False
            self.playing = False
            self.doctor = False

    def play_Click(self):
        if self.sleep_test():
            self.imageList = self.playImages
            self.playing = True
            self.walking = False
            self.eating = False
            self.doctor = False

    def walk_Click(self):
        if self.sleep_test():
            self.imageList = self.walkImages
            self.walking = True
            self.eating = False
            self.playing = False
            self.doctor = False

    def stop_Click(self):
        if not self.sleeping:
            self.imageList = self.nothingImages
            self.walking = False
            self.eating = False
            self.playing = False
            self.doctor = False

    def animation_timer(self):
        if self.sleeping and not self.forceAwake:
            self.imageList = self.sleepImages
        self.imageIndex += 1
        if self.imageIndex >= len(self.imageList):
            self.imageIndex = 0
        icon = QtGui.QIcon()
        current_image = self.imageList[self.imageIndex]
        icon.addPixmap(QtGui.QPixmap(current_image),
                       QtGui.QIcon.Disabled, QtGui.QIcon.Off)
        self.petPic.setIcon(icon)
        self.progressBar_1.setProperty("value", (8-self.hunger)*(100/8.0))
        self.progressBar_2.setProperty("value", self.happiness*(100/8.0))
        self.progressBar_3.setProperty("value", self.health*(100/8.0))

    def tick_timer(self):
        self.time_cycle += 1
        if self.time_cycle == 60:
            self.time_cycle = 0
        if self.time_cycle <= 48 or self.forceAwake:
            self.sleeping = False
        else:
            self.sleeping = True
        if self.time_cycle == 0:
            self.forceAwake = False
```

喂食按钮事件处理器

玩耍按钮事件处理器

散步按钮事件处理器

停止按钮事件处理器

动画定时器（第 0.5 秒）事件处理器

更新宠物的图像（动画）

5 秒定时器事件处理器开始

检查在睡觉还是醒着

```
                if self.doctor:
            self.health += 1
            self.hunger += 1
        elif self.walking and (self.time_cycle % 2 == 0):
            self.happiness += 1
            self.health += 1
            self.hunger += 1
        elif self.playing:
            self.happiness += 1
            self.hunger += 1
        elif self.eating:
            self.hunger -= 2
        elif self.sleeping:
            if self.time_cycle % 3 == 0:
                self.hunger += 1
        else:
            self.hunger += 1
            if self.time_cycle % 2 == 0:
                self.happiness -= 1
        if self.hunger > 8:  self.hunger = 8
        if self.hunger < 0:  self.hunger = 0
        if self.hunger == 7 and (self.time_cycle % 2 ==0) :
            self.health -= 1
        if self.hunger == 8:
            self.health -=1
        if self.health > 8:  self.health = 8
        if self.health < 0:  self.health = 0
        if self.happiness > 8:  self.happiness = 8
        if self.happiness < 0:  self.happiness = 0
        self.progressBar_1.setProperty("value",(8-self.hunger)*(100/8.0))
        self.progressBar_2.setProperty("value", self.happiness*(100/8.0))
        self.progressBar_3.setProperty("value", self.health*(100/8.0))

    def closeEvent(self, event):
        file = open("savedata_vp.pkl", "w")
        save_list = [self.happiness, self.health, self.hunger, \
                     datetime.datetime.now(), self.time_cycle]
        pickle.dump(save_list, file)
        event.accept()

    def menuExit_selected(self):
        self.close()

app = QtGui.QApplication(sys.argv)
myapp = MyForm()
myapp.show()
app.exec_()
```

根据活动增加或减少单位

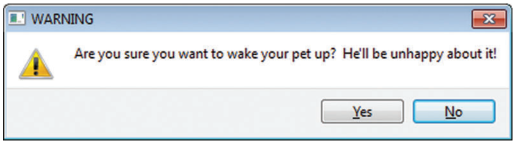
确保值没有越界

更新计量器

将状态和时间戳保存到 pickle 文件

行联接符

sleep_test() 函数使用了 PyQt 的“警告信息”对话框。一系列的参数告诉它要显示哪些按钮，以及哪个是默认按钮。代码清单 24-4 中的注释解释了这一点。当对话框弹出来时（你试图叫醒你的宠物时），看起来是这样的：



即使你不能完全读懂这个代码也不用担心。如果你希望学习更多有关 PyQt 的内容，可以先看看 PyQt 网站：www.riverbank computing.co.uk/software/pyqt/into。

在本章中，我们只是稍稍了解了计算机仿真的一点皮毛，知道了模拟真实世界中一些方面的基本思想，比如重力和时间。实际上，计算机仿真在科学、工程、医药、金融和很多其他领域都得到了广泛使用。其中很多仿真非常复杂，即使用最快的计算机运行也需要花费几天甚至几个星期。不过钥匙链上的小电子宠物也是一种仿真，有时最简单的仿真也是最有意思的。

你学到了什么

在这一章，你学到了以下内容。

- 什么是计算机仿真，为什么使用计算机仿真。
- 如何模拟重力、加速度和作用力。
- 如何跟踪和模拟时间。
- 如何使用 `pickle` 将时间戳保存到文件。
- 关于错误处理的一点知识（`try-except`）。
- 如何使用定时器生成周期性的事件。

测试题

1. 列出使用计算机仿真的 3 个原因。
2. 列出你见过或知道的 3 种计算机仿真。
3. 使用哪种对象来存储两个日期或时间之差？

动手试一试

1. 为 Lunar Lander 程序增加一个“脱离轨道”测试。如果飞船飞出了窗口顶边，而且速度超过 +100m/s，就停止程序，并显示一条消息，比如“You have escaped the moon’s gravity. No landing today!”（你已经脱离月球重力，无法着陆！）

2. 为 Lunar Lander 用户增加一个选项，可以在飞船着陆后继续玩这个游戏，而不必重启程序。
3. 为电子宠物 GUI 增加一个 Pause 按钮。这会让宠物的时间停止，不论程序是否在运行。（提示：这说明可能需要在 pickle 文件中保存“暂停”状态。）

第 25 章
Skier 游戏的说明

在第 10 章，你键入了 Skier 游戏的代码（希望如此）并运行了它。游戏代码中有一些注释，但除此之外我们没再给出别的说明。通常来说，键入代码然后运行它，是一种学习编程或学习一种语言的好方法，即使你并不能完全理解这些代码。

现在你对 Python 了解得更多了，可能会好奇这个 Skier 程序是怎么工作的。本章，我们会详细讲解这个程序。

25.1 滑雪者

首先，我们来编写游戏中的滑雪者。当你运行 Skier 程序的时候，应该注意到了滑雪者本身只能在屏幕上左右来回移动，而不能上下移动。滑雪者滑“下”小山的错觉则是通过将场景（树和小旗）向上滚动来实现的。

在滑雪者滑下小山的场景实现中，需要 5 张不同的图片：一张滑雪者一直向下滑，两张滑雪者向左转（一张稍向左转，一张大幅度左转），两张滑雪者向右转（一张稍向右转，一张大幅度右转）。在程序的开头，我们为这些图片创建了一个列表，然后将图片按特定的顺序放入列表。

```
skier_images = [ "skier_down.png",
                 "skier_right1.png","skier_right2.png",
                 "skier_left2.png","skier_left1.png"]
```

很快你就会知道为什么要按这样的顺序排放。

我们使用变量 `angle` 来标记滑雪者当前面对的方向，它的值从 −2 到 +2，分别如下：

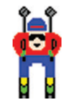
- −2 = 向左急转
- −1 = 稍向左转

- 0 = 一直向下
- +1 = 稍向右转
- +2 = 向右急转

（注意，这里的“左”和“右”是相对屏幕的方向，即我们看到的方向，而不是滑雪者的左和右。）

我们使用 angle 的值来确定要使用哪张图片。事实上，我们可以直接使用 angle 的值作为列表的索引来指定图片：

- skier_images[0] 是滑雪者向下滑的图片。



- skier_images[1] 是滑雪者稍向右转的图片。



- skier_images[2] 是滑雪者向右急转的图片。



接下来是微妙之处。还记得我们在第 12 章谈到过列表吗？我们说过负索引值会从列表的尾部开始往前数。所以，在这个例子中

- skier_images[-1] 是滑雪者稍向左转的图片（通常会使用 skier_images[4]）。



- skier_images[-2] 是滑雪者向左急转的图片（通常会使用 skier_images[3]）。



现在你知道为什么我们将列表中的图片按这种特定的顺序排列了：

- angle = +2（向右急转）= skier_images[2]
- angle = +1（稍向右转）= skier_images[1]
- angle = 0（向下滑）= skier_images[0]
- angle = −1（稍向左转）= skier_images[-1]（也就是 skier_images[4]）
- angle = −2（向左急转）= skier_images[-2]（也就是 skier_images[3]）

我们为滑雪者创建一个 Pygame Sprite 类的子类。滑雪者与窗口上边界的距离始终为 100 像素，开始时他位于窗口中心，也就是距离窗口左边界 320 像素，即

x=320，因为窗口的宽度是 640 像素。因此滑雪者的初始位置是 [320, 100]。滑雪者类定义的第一部分是下面这样：

```
class SkierClass(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image =
  pygame.image.load("skier_down.png")
        self.rect = self.image.get_rect()
        self.rect.center = [320, 100]
        self.angle = 0
```

我们用一个类来改变滑雪者的状态，它会在 angle 的值改变时载入正确的图片，并设置好滑雪者的速度。速度由 x 和 y 两个值构成。我们只改变左右方向的速度（x 方向的速度，x-speed），但是 y 方向的速度（y-speed）决定了场景向上滚动的速度（滑雪者向“下”滑的速度）。当他垂直向下运动时，y 方向的速度比较快，而当他转向时，y 方向的速度则比较慢。速度的计算公式如下：

```
speed = [self.angle, 6 - abs(self.angle) * 2]
```

此行代码中的 abs 用于取得 angle 的绝对值，也就是说我们忽略符号（+ 和 –）后的值。对于 y 方向的速度来说，滑雪者左转还是右转都没有影响，只要知道转的程度就行了。

用于转弯的全部代码如下：

```
def turn(self, direction):
        self.angle = self.angle + direction
        if self.angle < -2:  self.angle = -2
        if self.angle >  2:  self.angle =  2
        center = self.rect.center
        self.image = pygame.image.load(skier_images[self.angle])
        self.rect = self.image.get_rect()
        self.rect.center = center
        speed = [self.angle, 6 - abs(self.angle) * 2]
        return speed
```

我们还需要一个方法来控制滑雪者左右来回移动，以保证他不会滑出窗口边缘：

```
def move(self, speed):
        self.rect.centerx = self.rect.centerx + speed[0]
        if self.rect.centerx < 20:  self.rect.centerx = 20
        if self.rect.centerx > 620: self.rect.centerx = 620
```

我们使用方向键来控制滑雪者左右运动，所以要添加 Pygame 初始化和事件循环的代码，这样就可以让只有滑雪者的程序运行起来。代码如下所示。

代码清单 25-1 创建 Skier 游戏——只有滑雪者

```
import pygame, sys, random

skier_images = ["skier_down.png",
                  "skier_right1.png", "skier_right2.png",
                 "skier_left2.png", "skier_left1.png"]

class SkierClass(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("skier_down.png")
        self.rect = self.image.get_rect()
        self.rect.center = [320, 100]
        self.angle = 0

    def turn(self, direction):
        self.angle = self.angle + direction
        if self.angle < -2:  self.angle = -2
        if self.angle >  2:  self.angle =  2
        center = self.rect.center
        self.image = pygame.image.load(skier_images[self.angle])
        self.rect = self.image.get_rect()
        self.rect.center = center
        speed = [self.angle, 6 - abs(self.angle) * 2]
        return speed

    def move(self, speed):
        self.rect.centerx = self.rect.centerx + speed[0]
        if self.rect.centerx < 20:  self.rect.centerx = 20
        if self.rect.centerx > 620: self.rect.centerx = 620

def animate():
    screen.fill([255, 255, 255])
    screen.blit(skier.image, skier.rect)
    pygame.display.flip()

pygame.init()
screen = pygame.display.set_mode([640,640])
clock = pygame.time.Clock()
skier = SkierClass()
speed = [0, 6]

running  = True
while running:
    clock.tick(30)
    for event in pygame.event.get():
        if event.type == pygame.QUIT: running = False
        if event.type == pygame.KEYDOWN:
```

注释：
- 滑雪者的方向对应不同的图片（skier_images）
- 不让滑雪者的状态超过 +/-2（turn 中的两个 if 语句）
- 左右移动滑雪者（move 方法）
- 重绘屏幕（animate 函数）
- 检查按键事件（for event in pygame.event.get()）
- Pygame 主事件循环（while running）

```
            if event.key == pygame.K_LEFT:
                speed = skier.turn(-1)
            elif event.key == pygame.K_RIGHT:
                speed = skier.turn(1)
    skier.move(speed)
    animate()

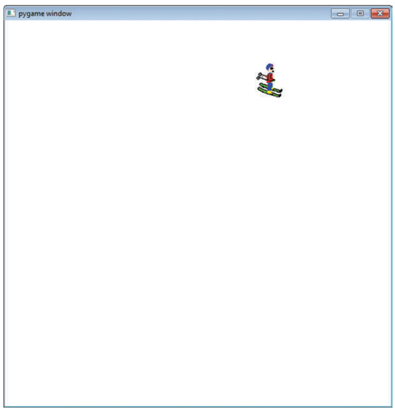
pygame.quit()
```

左方向键向左转

右方向键向右转

Pygame 主事件循环

如果运行以上代码，你只会在界面中看到滑雪者（没有分数，没有障碍物），你可以让他向左右转弯。



25.2　障碍物

接下来学习怎么制作障碍物，也就是树和小旗。在这一部分，为了简化起见，我们再次从零开始—没有滑雪者，只有障碍物。我们会在最后将滑雪者的代码和障碍物的代码放到一起。

Skier 游戏的窗口大小是 640×640 像素。为了简化，也为了防止障碍物靠得太近，我们将窗口分割为一个 10×10 的网格。这样，一共有 100 个格子，每个格子的大小是 64×64 像素。因为我们的障碍物尺寸并不是太大，所以即使两个障碍物位于相邻的格子中，它们之间也有一些空隙。

创建单个障碍物

首先我们需要创建单个障碍物。为此，我们创建了一个名为 `ObstacleClass` 的类。和滑雪者一样，这也是一个 Pygame `Sprite` 类。

```
class ObstacleClass(pygame.sprite.Sprite):
    def __init__(self, image_file, location, type):
        pygame.sprite.Sprite.__init__(self)
        self.image_file = image_file
```

```
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.center = location
        self.type = type
        self.passed = False
```

创建障碍物地图

现在我们来创建多个障碍物，以便填充在 640×640 像素的窗口中。我们将 10 个障碍物（小旗和树）随机地分布在 100 个格子中，每个障碍物既可以是小旗也可以是树。最终结果可能是 2 面小旗 8 棵树，7 面小旗 3 棵树，或者是总和为 10 的任意组合。总之，小旗和树是随机选择的，它们的位置也是随机的。

我们唯一需要当心的是，不要尝试将两个障碍物放在同一位置，所以我们需要知道哪些位置是已经使用过的。变量 `locations` 是一个用于记录使用过的位置的列表。当想要在某个位置放置一个新的障碍物时，我们首先要查看这个位置是否已有一个障碍物。

```
def create_map():
    global obstacles
    locations = []
    for i in range(10):
        row = random.randint(0, 9)
        col = random.randint(0, 9)
        location  = [col * 64 + 32, row * 64 + 32 + 640]
        if not (location in locations):
            locations.append(location)
            type = random.choice(["tree", "flag"])
            if type == "tree": img = "skier_tree.png"
            elif type == "flag":  img = "skier_flag.png"
            obstacle = ObstacleClass(img, location, type)
            obstacles.add(obstacle)
```

每屏 10 个障碍物（指向 `for i in range(10):`）

障碍物的位置 (x, y)（指向 `location = [...]`）

确保没有将两个障碍物放入同一位置（指向 `if not (location in locations):`）



好眼力！我们不希望游戏开始时满屏都是障碍物，而是希望它是空白的，然后障碍物从底部出现。所以，我们将障碍物的场景创建在窗口底部的“下方”。为此，我们需要为每个障碍物位置的 y 值增加 640 像素（窗口的高度）。

游戏开始时，我们希望障碍物从底部滚动上来。为此，我们修改每个障碍物位置的 y 值。改动的大小取决于滑雪者滑下小山的速度。我们将它放入一个名为 `update()` 的方法中，它是 `ObstacleClass` 的一部分。

```
    def update(self):
        global speed
        self.rect.centery -= speed[1]
```

变量 speed 是滑雪者的速度，它是一个全局变量，包含了 x 和 y 方向的速度，所以我们使用索引 [1] 来获取 y（垂直）方向的速度。

和创建的第一屏障碍物一样，我们还需要在窗口下方创建另外一屏的障碍物。那怎么知道应该在什么时候创建呢？我们可以创建一个名为 map_position 的变量，由它来告诉我们场景已经向上滚动了多少。我们在主循环中像下面这样处理。

```
running = True
while running:
    clock.tick(30)
    for event in pygame.event.get():
        if event.type == pygame.QUIT: running = False

    map_position += speed[1]          # 记录地图已经往上滚动了多少

    if map_position >= 640:           # 如果整屏已经滚动完，创建一个
        create_map()                  # 新的含障碍物的场景
        map_position = 0
```

我们用 animate() 函数来重绘屏幕，就像在只有滑雪者的代码中那样。合在一起，只有障碍物的代码看起来像下面这样。

代码清单 25-2　创建 Skier 游戏——只有障碍物

```
import pygame, sys, random

class ObstacleClass(pygame.sprite.Sprite):                 # 障碍物的类（树和小旗）
    def __init__(self, image_file, location, type):
        pygame.sprite.Sprite.__init__(self)
        self.image_file = image_file
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.center = location
        self.type = type
        self.passed = False

    def update(self):
        global speed
        self.rect.centery -= speed[1]

def create_map():                                           # 创建一个有障碍物的场景：640 × 640
    global obstacles
    locations = []
    for i in range(10):                                     # 每屏 10 个障碍物
        row = random.randint(0, 9)
        col = random.randint(0, 9)
```

```
        location  = [col * 64 + 32, row * 64 + 32 + 640]
        if not (location in locations):
            locations.append(location)
            type = random.choice(["tree", "flag"])
            if type == "tree": img = "skier_tree.png"
            elif type == "flag":  img = "skier_flag.png"
            obstacle = ObstacleClass(img, location, type)
            obstacles.add(obstacle)

def animate():
    screen.fill([255, 255, 255])
    obstacles.draw(screen)
    pygame.display.flip()

pygame.init()
screen = pygame.display.set_mode([640,640])
clock = pygame.time.Clock()
speed = [0, 6]
obstacles = pygame.sprite.Group()
map_position = 0
create_map()

running = True
while running:
    clock.tick(30)
    for event in pygame.event.get():
        if event.type == pygame.QUIT: running = False

    map_position += speed[1]

    if map_position >= 640:
        create_map()
        map_position = 0

    obstacles.update()
    animate()

pygame.quit()
```

防止 2 个障碍物放在同一位置

重绘屏幕

初始化

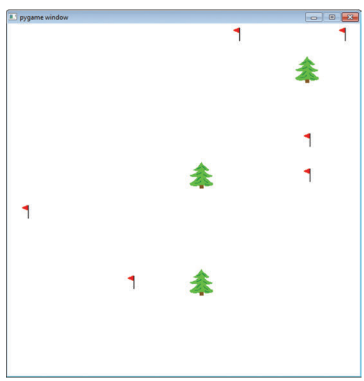
主循环

记录地图已经往上滚动了多少

在下面创建一个新的含障碍物的场景

主循环

如果运行以上代码，你应该可以看到树和小旗在屏幕上往上滚。





问得好。在当前的代码中，我们让它们继续在窗口上边界外向上滚动，这样其位置的 y 值（负值）绝对值会越来越大。如果游戏运行了比较长的时间，则会创建并积累大量的障碍物场景。这有可能会导致程序变慢或者在某个时间点发生内存不足的情况。所以我们需要做一点清理工作。

在障碍物类的 update() 方法中，我们添加一个判断逻辑，看看障碍物是否已经移出屏幕。如果是的话，就移除它。Pygame 有一个名为 kill() 的原生方法可用来做这件事。新的 update() 方法看起来像下面这样。

```
def update(self):
        global speed
        self.rect.centery -= speed[1]
        if self.rect.centery < -32:        ← 检查障碍物是否已移出屏幕
            self.kill()                    ← 移除它
```

现在我们可以将滑雪者和障碍物的代码放到一起了。

- 我们需要 SkierClass 和 ObstacleClass。
- 我们的 animate() 函数需要同时绘制滑雪者和障碍物。
- 我们的初始化代码需要创建滑雪者和初始地图。
- 主循环需要同时包括滑雪者的键盘事件绑定和障碍物场景的创建。

基本上，这组合了代码清单 25-1 和代码清单 25-2，结果如下所示。

代码清单 25-3　滑雪者代码与障碍物代码相结合

```
import pygame, sys, random

skier_images = ["skier_down.png", "skier_right1.png", "skier_right2.png",
                 "skier_left2.png", "skier_left1.png"]

class SkierClass(pygame.sprite.Sprite):
    def __init__(self):
```

```
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("skier_down.png")
        self.rect = self.image.get_rect()
        self.rect.center = [320, 100]
        self.angle = 0

    def turn(self, direction):
        self.angle = self.angle + direction
        if self.angle < -2:  self.angle = -2
        if self.angle >  2:  self.angle =  2
        center = self.rect.center
        self.image = pygame.image.load(skier_images[self.angle])
        self.rect = self.image.get_rect()
        self.rect.center = center
        speed = [self.angle, 6 - abs(self.angle) * 2]
        return speed

    def move(self, speed):
        self.rect.centerx = self.rect.centerx + speed[0]
        if self.rect.centerx < 20:  self.rect.centerx = 20
        if self.rect.centerx > 620: self.rect.centerx = 620
```

滑雪者代码

```
class ObstacleClass(pygame.sprite.Sprite):
    def __init__(self, image_file, location, type):
        pygame.sprite.Sprite.__init__(self)
        self.image_file = image_file
        self.image = pygame.image.load(image_file)
        self.rect = self.image.get_rect()
        self.rect.center = location
        self.type = type
        self.passed = False

    def update(self):
        global speed
        self.rect.centery -= speed[1]
        if self.rect.centery < -32:
            self.kill()

def create_map():
    global obstacles
    locations = []
    for i in range(10):
        row = random.randint(0, 9)
        col = random.randint(0, 9)
        location  = [col * 64 + 32, row * 64 + 32 + 640]
        if not (location in locations):
            locations.append(location)
            type = random.choice(["tree", "flag"])
            if type == "tree": img = "skier_tree.png"
            elif type == "flag":  img = "skier_flag.png"
            obstacle = ObstacleClass(img, location, type)
            obstacles.add(obstacle)
```

障碍物代码

```
def animate():
    screen.fill([255, 255, 255])
    obstacles.draw(screen)
    screen.blit(skier.image, skier.rect)
    pygame.display.flip()

pygame.init()
screen = pygame.display.set_mode([640,640])
clock = pygame.time.Clock()
points = 0
speed = [0, 6]
skier = SkierClass()
obstacles = pygame.sprite.Group()
create_map()
map_position = 0

running = True
while running:
    clock.tick(30)
    for event in pygame.event.get():
        if event.type == pygame.QUIT: running = False

        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                speed = skier.turn(-1)
            elif event.key == pygame.K_RIGHT:
                speed = skier.turn(1)
    skier.move(speed)

    map_position += speed[1]

    if map_position >= 640:
        create_map()
        map_position = 0

    obstacles.update()
    animate()

pygame.quit()
```

重绘滑雪者和障碍物

创建滑雪者

创建障碍物

初始化

主循环

主循环

如果运行以上代码，你就能操作滑雪者滑下小山，并且会看到障碍物向上滚过。你也会注意到滑雪者向左右滑和向下滑的速度取决于他的转向。至此，这个游戏已经快要完成了。

最后我们需要处理的两项是：

- 检测滑雪者是否碰到了树或者是否捡到了小旗
- 记录并显示分数

你已经在第 17 章中学到如何进行碰撞检测了。代码清单 25-3 已经将障碍物精

灵放到了一个 sprite 组中，所以我们可以直接用 `spritecollide()` 函数来检测滑雪者是否碰到树或者小旗。接下来我们需要知道这个障碍物是什么（树还是小旗），然后：



- ❒ 如果是树，则将滑雪者的图像切换为“碰撞”的图像，并将分数减去 100。
- ❒ 如果是小旗，则将分数加 10，然后将小旗从屏幕中移除。

为此所需的代码位于主循环中，看起来像下面这样：

```
hit =  pygame.sprite.spritecollide(skier, obstacles, False)   <- 碰撞检测
    if hit:
        if hit[0].type == "tree" and not hit[0].passed:       | 碰到树
            points = points - 100                             |


            skier.image = pygame.image.load("skier_crash.png")  | 显示碰撞
            animate()                                           | 后的图像
            pygame.time.delay(1000)                             |
            skier.image = pygame.image.load("skier_down.png")   | 继续滑雪
            skier.angle = 0                                     |
            speed = [0, 6]                                      |
            hit[0].passed = True        <- 别忘了我们已经碰到了这棵树

        elif hit[0].type == "flag" and not hit[0].passed:     | 捡到小旗
            points += 10                                      |
            hit[0].kill()         <- 移除小旗
```

变量 `hit` 告诉我们滑雪者撞到了哪个障碍物。它是一个列表，但在这里列表中只有一个条目，因为滑雪者一次只可能碰到一个障碍物，所以他碰到的障碍物是 `hit[0]`。

`passed` 变量用于标记是否碰撞过树。这能保证当滑雪者在撞树之后继续往下滑时，不会立刻运行撞到同一棵树的逻辑。

现在我们需要显示分数。这只需要再写 3 行代码。在初始化代码中，我们创建一个 `font` 对象，它是 Pygame 的 `Font` 类的实例：

```
font = pygame.font.Font(None, 50)
```

在主循环中，我们使用一个新的分数文本来渲染 `font` 对象：

```
score_text = font.render("Score: " +str(points), 1, (0, 0, 0))
```

在 `animate()` 函数中，我们在左上角显示分数：

```
screen.blit(score_text, [10, 10])
```

这就是这个游戏的全部了。如果你把这些组合在一起，会发现这就是第 10 章的

代码，只是现在你理解得更加深入了。理解 Skier 游戏的工作原理可以帮助你更好地策划和开发自己的游戏。

你学到了什么

在这一章，你学到了以下内容。

- Skier 游戏的各个部分是如何工作的
- 怎样创建一个滚动的背景

动手试一试

1. 尝试修改 Skier 游戏，使游戏的难度随着游戏的进行逐渐增大。可以尝试以下建议：
   - 使游戏的速度随着游戏的进行逐渐加快。
   - 滑雪者滑下小山时树越来越多。
   - 添加障碍物“冰”，使得滑雪者转向时更加困难。
2. Skier 游戏的灵感来自一个名为 SkiFree 的游戏，这个游戏中有一个讨厌的雪人会随机出现并追赶滑雪者。如果你想挑战一下，可以尝试在 Skier 游戏中添加一些类似的物体。你需要找到或者创建一个新的图片，并且修改代码以获得你想要的行为。

第 26 章 Python Battle

在本书中我们已经讲解了如何开发自己的游戏，但还有一个话题没有讨论，那就是游戏中的人工智能（AI）。从吃豆人（Pac-Man）开始，几乎所有的游戏在攻击玩家时都有某种形式的人工智能。本章将展示如何动手开发一个有人工智能的游戏。

26.1 Python Battle

在本章中我们要为一个名为 Python Battle 的游戏开发 AI。Python Battle 是一个规则很简单的游戏。在每一回合中，你可以向前移动、向左右转或者攻击对方。当一个角色攻击另一个角色时，被攻击的一方会减少一点“血量”。血量先降为 0 者输掉游戏。角色只可以攻击正前方。

但 Python Battle 有一个特殊之处：它是由机器人玩的游戏。你必须要编码一个控制机器人的策略，或者叫 AI，然后运行游戏，看看它和另一个 AI 对抗的效果如何。当然，AI 部分的代码和 Python Battle 游戏本身的代码都使用 Python 来编写。

注意，如果你之前运行了本书的安装程序，那么你的计算机中应该已经有 Python Battle 的代码了。我放了三个 AI 程序，你可以一一尝试：CircleAI（在游戏区域中转圈）、RandomAI（随机移动和转向）和 NullAI（什么也不做）。我们来一一运行一下，看看谁会赢。

运行 Python Battle

按以下步骤运行。

(1) 将你想运行的 AI 脚本放到与 PythonBattle.py 相同的目录下。
(2) 运行 PythonBattle.py。
(3) 你会看见一个提示框：

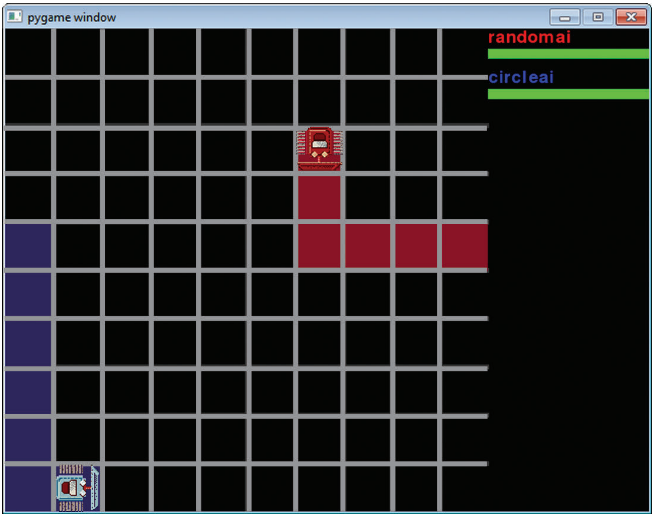
`Enter red AI:`

输入你想使用的 AI 脚本的名字，结尾不用带 .py 后缀。例如，如果你想测试 CircleAI.py，就输入 `circleai`。

(4) 针对蓝色的 AI 进行同样的操作。

(5) 观察游戏，看看谁会赢。

(6) 最后关掉 Pygame 窗口。



游戏规则

现在你应该看过了好几场 Python Battle 了，我们来仔细看一下游戏是怎么运行的。每一回合，机器人都可以做以下 6 件事之一：

- ❒ 向前移动一格
- ❒ 向后移动一格
- ❒ 左转
- ❒ 右转
- ❒ 攻击正前方的格子
- ❒ 什么都不做

此外，机器人可以随时获取双方的血量。游戏的目标是成功攻击敌人 10 次。

你在观察 CircleAI 和 RandomAI 对抗时，有没有留意红色和蓝色的方块？当机器人经过一个方块时，这个方块会变成和机器人一样的颜色（红色或者蓝色）。如果 1000 个回合之后还没有机器人成功攻击对方，或者双方打成了平局，则拥有和自己颜色一致的方块较多者获胜。

26.2 创建一个 Python Battle 机器人

我们来创建一个机器人，以便与 CircleAI 和 RandomAI 战斗。第一步是制定一个策略。因为这是我们创建的第一个 Python Battle 机器人，所以我们最好从一个简单的策略开始。

(1) 如果面对着一个敌人，那么我应该攻击它。
(2) 如果面对着墙，则我应该转弯。
(3) 否则，我继续向前走。

尽管这不是最高级的策略，但它可能刚好可以打败 CircleAI。如果它打不过，我们还可以再回来改进它。

第二步是开始编码机器人。创建一个新 Python 文件（我把它叫作 better_than_circleai.py），然后输入以下代码：

```
class AI:
    def __init__(self):
        pass
    def turn(self):
        pass
```

这是所有机器人都必须具备的基础代码。在游戏开始时创建机器人，此时 `__init__()` 函数会被调用。`turn()` 函数会在每一回合被调用，并决定机器人做什么。这个类必须叫作 AI，否则 Python Battle 游戏不知道你的 AI 代码在哪里。

编写机器人的下一步是添加能让机器人执行我们制定的策略的代码。你将使用以下函数来控制机器人的移动：

- ❒ `self.robot.lookInFront()`
- ❒ `self.robot.turnRight()`
- ❒ `self.robot.turnLeft()`
- ❒ `self.robot.goForth()`
- ❒ `self.robot.attack()`

这些函数需要采用 `self.robot.xx()` 的形式调用，因为 self 指向 AI 对象本身，而不是 AI 控制的机器人。AI 对象是不会移动或者攻击的，它只负责告知机器人需要做什么，具体的移动和攻击是由机器人对象来执行的。

我们来编写 `turn()` 方法。移除代码中的 pass，添加下面的代码：

```
self.robot.goForth()
```

如果你现在测试一下这个机器人，会看到每一回合它都会尝试向前移动。这会

导致它很快撞到游戏边界处的“墙”。如果敌人挡住了它的路，则它只会停止移动而已。你可以使用 self.robot.lookInFront() 来修复这个问题。如果在正前方有一个可以攻击的敌人，则这个函数返回 "bot"。修改一下 turn() 函数，像这样：

```
if self.robot.lookInFront() == "bot":
    self.robot.attack()
else:
    self.robot.goForth()
```

现在，如果在你的机器人前方有敌人挡路，则机器人会攻击它。但是如果机器人撞“墙”了，则它会停下来。如果前方是“墙”，则 self.robot.lookInFront 会返回 "wall"。将下面的代码加到 turn() 函数中，放在 self.robot.attack() 和 else 之间：

```
elif self.robot.lookInFront() == "wall":
    self.robot.turnRight()
```

现在，如果你运行代码，则机器人会开始转圈！当它撞到“墙”的时候会向右转，当它撞到下一堵“墙”时会再次向右转，一直这样运行下去。你更希望机器人在撞“墙”的时候能转身折返回去，而这需要两次右转才能完成。当机器人撞“墙”的时候，它需要右转一次，然后在一下回合中，再右转一次。也就是说，你需要机器人记住它现在是在做 U 型转身。你可以给 AI 设置一个新属性（变量）来记住它正在做什么。我们在 __init__() 函数中添加一行代码：

```
self.currentlyDoing = "forward"
```

这将告诉机器人，当游戏开始时，它需要向前走。当机器人在做 U 型转身时，你可以将这个属性的值改为 "turnRight" 以便告知机器人它需要右转。修改之后的 turn() 函数最终看起来像这样：

代码清单 26-1　完成的机器人 AI

```
class AI:
    def __init__(self):
        self.currentlyDoing = "forward"
    def turn(self):
        if self.robot.lookInFront() == "bot":
            self.robot.attack()
        elif self.robot.lookInFront() == "wall":
            self.robot.turnRight()
            self.currentlyDoing = "turnRight"
        elif self.currentlyDoing == "turnRight":
            self.robot.turnRight()
            self.currentlyDoing = "forward"
        else:
            self.robot.goForth()
```

如果你运行这段代码，可能会发现这个 AI 有一定的缺陷。我们来修改一下以便我们的机器人可以打败 CircleAI。

26.3　更复杂的机器人

我们的第一个机器人非常简单，它无法战胜 CircleAI。为了战胜其他所有的机器人，我们需要一个真正优秀的策略。一个真正优秀的策略绝不止“转圈圈，直到敌人出现在我眼前”这么简单，而是需要用到每一个可用的命令。最重要的是，真正优秀的策略需要我们花费大量心思来考虑它的工作机制。

还有一些方法我在上一节没有提到，这些方法可以帮助我们制定一个必胜的策略。

- `self.robot.goBack()` ——顾名思义，机器人可以后退一步。
- `self.robot.checkSpace(space)` ——可以检查任意一个方块。例如，`self.robot.checkSpace((3,3))` 可以返回 (3,3) 这个方块处有什么。如果什么也没有，则返回 `"blank"`，否则，返回 `"bot"`（如果有敌人）、`"me"`（如果有机器人）或者 `"wall"`（如果这个方块在游戏边界之外）。
- `self.robot.locateEnemy()` ——返回敌人的位置和方向。
- `self.robot.position` ——获取机器人的位置。
- `self.robot.rotation` ——获取机器人的方向。
- `self.robot.calculateCoordinates(direction, distance, position)` ——稍后解释。首先，我们需要理解在 Python Battle 中，坐标系统是怎么工作的。

26.4　坐标系统

在 Python Battle 中，坐标系统的范围从 (1,1) 到 (10,10)。和之前的 Pygame 一样，坐标的原点在左上角。游戏区域的四面都被墙围住。我们可以用 `self.robot.position` 来找到机器人在坐标系统中的位置。

方向

使用数字 0~3 来存储方向。0 是上（北），1 是右（东），2 是下（南），3 是左（西）。当机器人右转时，方向的值加 1；当机器人左转时，方向的值减 1。这样用起来很简单。我们可以使用 `self.robot.rotation` 来获取机器人的方向。

`calculateCoordinates()`

`calculateCoordinates()` 函数接受三个参数：`distance`、`direction` 和 `position`。它用于查找在 `direction` 方向上离 `position` 距离为 `distance` 的方块。

例如，calculateCoordinates(2,3,(5,5)) 用于查找距离 (5,5) 左边（左边方向值为 3）2 个方块远的方块。

现在我们可以想出一个策略。我采用一个很简单的策略：

(1) 向着敌人所在的方向前进。
(2) 如果可能，则发动攻击。

我们从前一个机器人的一些基础代码开始：

```
class AI:
    def __init__(self):
        pass
    def turn(self):
        if self.robot.lookInFront() == "bot":
            self.robot.attack()
```

这段代码会处理我们策略的第二部分：“如果可能，则发动攻击。”现在我们需要处理第一部分。我们将下面的代码加到 turn() 函数中：

```
else:
     self.goTowards(self.robot.locateEnemy()[0])
```

这会调用 AI 类的方法 self.goTowards() 并传入敌人的位置作为参数。self.robot.locateEnemy() 方法会返回一个包含敌人位置和方向的列表。如果你运行这段代码，它不会正常工作，因为我们还没有定义 self.goTowards()。现在我们来定义它：

```
def goTowards(self,enemyLocation):
     myLocation = self.robot.position
     delta = (enemyLocation[0]-myLocation[0],
              enemyLocation[1]-myLocation[1])
```

首先算出 delta，即目标位置和机器人位置的差距。然后，你需要知道要想面向敌人，机器人应该面向哪个方向：

```
if abs(delta[0]) > abs(delta[1]):
     if delta[0] < 0:
         targetOrientation = 3        # 面向左
     else:
         targetOrientation = 1        # 面向右
else:
    if delta[1] < 0:
        targetOrientation = 0         # 面向上
    else:
        targetOrientation = 2         # 面向下
```

现在你需要沿着这个方向走。如果已经面向这个方向的话，则很简单：

```
if self.robot.rotation == targetOrientation:
    self.robot.goForth()
```

否则，你需要找到应该往哪个方向转。首先，你需要知道如果要转到正确的方向，需要左转多少次：

```
else:
     leftTurnsNeeded = (self.robot.rotation - targetOrientation) % 4
```

接下来，需要转到正确的方向。如果需要左转 2 次以上，则可以只右转 1 次：

```
if leftTurnsNeeded <= 2:
     self.robot.turnLeft()
else:
     self.robot.turnRight()
```

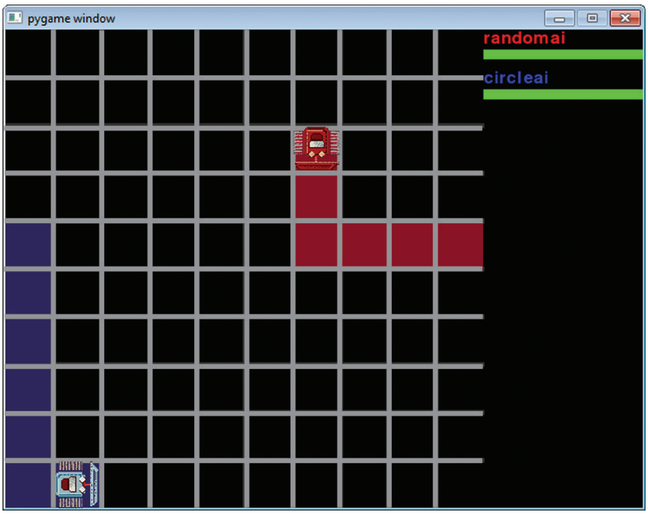
以下是机器人的完整代码。

代码清单 26-2　更复杂的机器人

```
class AI:
    def __init__(self):
        pass
    def turn(self):
        if self.robot.lookInFront() == "bot":
            self.robot.attack()
        else:
            self.goTowards(self.robot.locateEnemy()[0])
    def goTowards(self,enemyLocation):
        myLocation = self.robot.position
        delta=(enemyLocation[0]-myLocation[0],enemyLocation[1]-myLocation[1])
        if abs(delta[0]) > abs(delta[1]):
            if delta[0] < 0:
                targetOrientation = 3          # 面向左
            else:
                targetOrientation = 1          # 面向右
        else:
            if delta[1] < 0:
                targetOrientation = 0          # 面向上
            else:
                targetOrientation = 2          # 面向下
        if self.robot.rotation == targetOrientation:
            self.robot.goForth()
        else:
            leftTurnsNeeded = (self.robot.rotation - targetOrientation) % 4
            if leftTurnsNeeded <= 2:
                self.robot.turnLeft()
            else:
                self.robot.turnRight()
```

现在，我们来试着挑战一下 CircleAI。我确信我们的辛苦工作会得到回报，我们一定能打败它。将 AI 保存为 morecomplicatedai.py，然后重新运行 Python Battle：

```
>>>
Enter red AI: circleai
Enter blue AI: morecomplicatedai
    .
    .
    .
Red wins with 10 health!
```





我想我们需要稍微修订一下策略以便打败 CircleAI。CircleAI 这么难打败，主要是因为我们很难打到它。如果你打算从侧面或者后面攻击它，那么在你第二次打到它之前它就会跑掉。同时，因为它靠墙走，所以你只有一个侧面可以攻击。如果你从前面攻击的话，它就很可能拿到第一次攻击的机会并最终获胜。尽管 CircleAI 不是最高级的策略，但它确实很难打败。

你也许能设计出一个可以稳妥打败 CircleAI 的 AI，但前提是你知道自己要对付的 AI 是 CircleAI。如果你不知道对方 AI 将要使用的策略，打败它将更加困难！

你学到了什么

在这一章，你学到了以下内容。

- 游戏是怎么运用 AI 让敌人变聪明的。
- 如何在 Python Battle 游戏中创建自己的 AI。

动手试一试

修订我的策略，尝试设计出一个能打败 CircleAI 的机器人。

第 27 章

接下来呢

本书已接近尾声。如果你读完了整本书，并且尝试过本书里的所有例子，现在应该对编程以及利用编程能够做什么已经有了基本的了解。

本章会告诉你可以在哪里查找关于编程的更多信息。有很多资源可以利用，有些关于一般编程，有些专门针对 Python 编程，还有一些关于游戏编程以及其他一些方面。

如何进一步学习编程，这要看你想用它做什么。你已经从 Python 起步，在这本书中学到的很多东西都是一般性的编程思想和概念，在其他计算机语言中也完全适用。如何学以及学些什么取决于你想在哪个方向深入：游戏？ Web 编程？还是机器人？（机器人需要软件来告诉它们做什么。）

27.1　致年龄小的读者

对年龄小的读者来说，如果你喜欢用 Python 学习编程，可能也会乐于尝试另一种方法。Squeak Etoys 是一种面向孩子们的编程“语言”，它几乎是完全图形化的。你几乎不用写任何代码，可以通过创建图形对象并修改它们的属性和动作来建立程序。在后台，这些图形对象会转换为一种 Smalltalk 语言的代码，可以在 squeakland.org 了解更多有关 Etoys 的内容。

另一种看起来和 Squeak 很像的语言是 Scratch。与 Squeak 类似，Scratch 也允许你使用拖拽的方式编程。Scratch 程序也可以放到 Web 上共享。你可以在 scratch.mit.edu 上获取 Scratch。

27.2　Python

很多地方都可以帮助你更深入地学习 Python。在线 Python 文档非常完备，不过读起来可能有点困难。它包含一个语言参考、库参考、全局模块索引和 Guido van Rossum 写的一个教程（正是他创建了 Python）。你可以在这里找到这个文件：docs.python.org。

市面上有很多关于 Python 高阶内容的书籍，多到我无法只向你推荐一两本。具体使用什么书取决于你的品味、学习方式以及你想用 Python 做的事情。但我坚信，如果你想在 Python 学习之路上走得更远，就一定能找到适合自己的书。

邮件列表也非常有用。你可以发布一个消息，其他用户就会尽力来回答你的问题。大多数列表都有归档页面，你可以阅读或搜索较早的消息，看看是不是已经有人问过你要问的问题。

27.3　游戏编程与 Pygame

如果你只是想建立游戏，关于这个主题有很多书，实在是太多了，根本无法在这里一一列出。你可能想学习一种 OpenGL 技术，这是“Open Graphics Language”（开放图形语言）的简写，很多游戏都使用了这种图形系统。在 Python 中可以使用一个名为 `PyOpenGL` 的模块来使用 OpenGL，关于这个内容也有很多书可以参考。

如果你对 Pygame 感兴趣，也可以找到一些地方来了解更多有关内容。Pygame 网站（pygame.org）提供了很多例子和教程。

如果你确实想用 Pygame 完成游戏编程，向你推荐两个非常棒的资源。一个是 Pygame 邮件列表。我发现这个资源很有用。你可以在 pygame.org/wiki/info 找到它。邮件列表地址是 pygame-users@seul.org。

如果你想在游戏中实现更逼真的物理行为，可以使用一个名为 `PyMunk` 的库。`PyMunk` 是基于 Chipmunk Physics 开发的。你可以利用 Chipmunk 在二维（2D）世界中创建圆、直线和图形等，然后它会让这些图形模拟出重力和摩擦力等物理学上基本的力。可以在 pymunk.org 获取 `PyMunk`。

27.4　其他语言的游戏编程（非 Python）

我如果你对游戏编程感兴趣，可能会有兴趣了解 Unity 游戏引擎。Unity 包含了很多东西，包括一个 3D 游戏引擎和一个物理引擎，并提供了编写脚本的方式。你可以用来编写脚本的语言之一叫做 Boo，它与 Python 有很多相似之处。

你可能玩过一些游戏，现在你可以通过编写代码来扩展它们。比如，要扩展 Roblox（www.roblox.com），你可以编写 Lua 代码。你也可以在 Minecraft（www.minecraft.com）中使用 Lua 或者 Java 编写代码。（顺便提一下，流行的 Angry Birds 游戏就是使用 Lua 编写的。）

27.5　传承 BASIC

你可能注意到这样一种现象，如果在图书馆找书，可以找到 20 世纪 80 年代为孩子们写的一些编程书，而且其中很多书都使用了一种名为 BASIC 的语言，这在当时相当流行。（现在你还能得到面向现代计算机的一些 BASIC 版本，包括面向 Windows 的 QBASIC 和 BBC BASIC。）这些书里往往有很多游戏。如果把这些古老的 BASIC 书中的游戏用 Python 重写可能很有意思。如果需要，你可以使用 Pygame 或 PythonCard 来帮助完成图形部分。我保证这样会让你大有收获！

27.6　移动应用

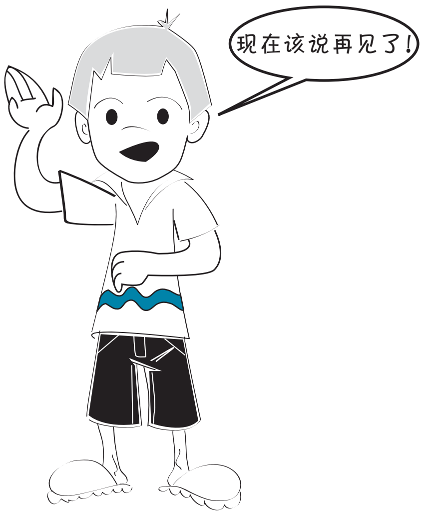
如果你对编写 iPhone 或者 Android 应用感兴趣的话，可以通过以下两种方式来编写。你可以通过 PhoneGap（phonegap.com）使用 HTML5 和 CSS 来编写 iPhone 应用。如果采用这种方式，你就很容易将应用移植到其他的手机操作系统上（如 Android 或者 Blackberry OS）。另一种方式是使用 Objective-C 语言和 Cocoa Touch 库编写原生的 iPhone 应用代码。编写原生代码使用的编辑器叫做 Xcode，它只能在 Mac OS X 上运行。Android 应用基本上是用 Java 编写的，但你也可以利用像 PhoneGap 这样的库来使用其他语言编写。

27.7　回顾

有很多很多别的主题需要研究，还有很多资源可以帮助你在不同的编程领域尤其是 Python 编程领域中更为深入。你可以在图书馆或书店找一找，看看哪些书提供了你感兴趣的信息。也可以在网上搜索这些主题，看看有没有一些在线教程或者 Python 模块能帮你达成目标。

利用 Python 可以做很多事，但如果你需要做一些特定的事情，可能需要其他的语言，比如 C、C++、Java、JavaScript（和 Java 不同）或者别的什么语言。如果是这样的话，你就需要找到一本书或者其他资源来学习这门语言。这些书和资源非常之多，所以我也无法在这个话题上给出更多的建议了。

不管怎样，享受编程的快乐吧！不断学习、探索和试验。你对编程了解得越多，就会发觉它越有意思！



附录 A
变量命名规则

下面是关于变量名（也称为标识符）的一些规则。

- 必须以一个字母或一个下划线字符开头。后面可以使用一个字母、数字或下划线字符的序列，长度不限。
- 字母可以是大写或小写，大小写是不同的。也就是说，`Ax` 不同于 `aX`。
- 数字可以是从 0 到 9（包括 0 和 9）的任意数字字符。

除了字母、数字和下划线字符，不能使用其他字符。空格、标点符号和其他字符在变量名中都是不允许的：

```
~ ` ! @ # $ % ^ & * ( ) ; - : " ' < > , . ? / { } [ ] + = /
```

唯一允许出现的特殊字符是下划线字符。也许你不知道这是什么，下面给出几个例子：

- `first_number = 15`
- `student_name = "John"`

`first` 和 `number` 之间的字符就是下划线。另外在 `student` 和 `name` 之间也有一个下划线。程序员有时会使用下划线分隔变量名中的两个单词。因为空格在变量名中是不允许的，所以他们会使用下划线。

建议你不要在变量名开始和末尾使用下划线字符，除非你很清楚为什么要这样做。有些情况下，在一个标识符开始和末尾使用下划线字符会有特殊的含义。所以要避免这样使用：

- `_first_number = 15`
- `student_name_ = "John"`

下面是一些合法变量名的例子：

- my_answer
- answer23
- answer_23
- YourAnswer
- Your2ndAnswer

下面是一些不合法变量名的例子：

- 23answer（变量名不能以数字开头。）
- your-answer（不允许有连字符。）
- my answer（不允许有空格。）

附录 B
Python 2 与 Python 3 的差异

在本书中，我们提到了 Python 2 与 Python 3 的几处差异。本书使用的是 Python 2，但我们也希望你能读懂 Python 3 的代码，并具备让代码兼容 Python 3 的能力。本附录只讨论与本书内容有关的 Python 特性的差异。（比如，Python 3 处理 Unicode 字符串的方式是不同的，但因为我们在本书中没有涉及 Unicode，所以不讨论这部分差异。）

在这个前提下，Python 3 与 Python 2 存在以下一些差异。

print

在 Python 3 中，`print` 是一个函数，这意味着你不能这样写：

```
print "Hello, World!"
```

而需要这样写：

```
print("Hello, World!")
```

还有一些与此相关的差异。你不能像在 Python 2 中那样，在结尾处使用一个逗号，使下一个 `print` 语句输出的内容与本次输出的内容显示在同一行：

```
print "Hello",
print "World!"
```

在 Python 3 中，你需要这样做：

```
print("Hello", end='')
print("World!")
```

你也可以给 `end` 参数传不同的值，但是用到的情况非常少。如果你有兴趣的话，可以查一下 Python 3 的文档。

你可能已经注意到了，print 在 Python 3 和 Python 2 代码中的高亮颜色是不一样的。这是因为在 Python 2 中，print 是一个关键字，而在 Python 3 中，print() 是一个函数。

input()

Python 2 中的 raw_input() 在 Python 3 中改名为 input() 了。这个函数会返回字符串，但不会尝试对输入的字符串做任何计算（比如将它转成 int 或 float）。

Python 2 中的 input() 函数会对输入进行计算（如果可能的话会将它转换成数字），这个函数在 Python 3 中不存在了。

这意味着在 Python 2 中的代码

```
your_name = raw_input("Enter your name:")
```

在 Python 3 中要这样写：

```
your_name = input("Enter your name:")
```

在 Python 2 中的代码

```
age = input("Enter your age:")
```

在 Python 3 中要这样写：

```
age = int(input("Enter your age:")
```

整除

Python 3 中第三点主要的变化是处理整除的方式。是否还记得我们在第 3 章讨论过这个话题？在 Python 2 中，代码运行结果是这样的：

```
>>> print 5/2
2
```

Python 2 在进行除法运算时会默认向下取整。Python 3 会默认进行浮点数除法，所以在 Python 3 中是这样的：

```
>>> print(5/2)
2.5
```

如果你想在 Python 3 中进行整除运算，可以使用双斜线，像这样：

```
>>> print(5//2)
2
```

取余操作符（%）用于计算整数相除的余数，其在 Python 3 中的工作方式和在 Python 2 中一样。

Python 2：
```
>>> print 5%2
1
```
Python 3：
```
>>> print(5%2)
1
```

range()

在 Python 2 中，range() 函数返回一个列表。但在 Python 3 中，range() 返回一个 range 对象。如果是循环访问的话，Python 3 中 range 对象的工作方式和 Python 2 中的列表基本一样。但如果你想打印它，是不能看到 range 对象中的各个单独的值的。

Python 2：
```
>>> print range(5)
[0, 1, 2, 3, 4]
```
Python 3：
```
>>> print(range(5))
range(0, 5)
```

Python 2 到 Python 3 的转换

有一个名为 2to3 的工具会尝试自动将 Python 2 代码转换为 Python 3 代码。如果你在使用 Python 3，可以拿本书中的代码清单试一下。很多代码在转换后都能正常工作。但我们没有测试本书中的代码经 2to3 转换后能否正常工作，所以我们不保证它适用于本书所有的代码清单。

尽管 Python 3 是 Python 的最新版本，但 Python 2 现在还是被广泛支持。因为非常非常多的人编写的 Python 2 代码不能在 Python 3 中正常工作，所以很多公司和个人都继续使用 Python 2。有很多模块还没有升级，无法支持 Python 3，甚至有一些模块从来没有升级过。Python 2 的资源要远比 Python 3 的多，这也是我们决定在本书中继续使用 Python 2 的一个原因。但如果你想尝试在自己的编程项目中使用 Python 3，我们祝你好运。

附录 C

自测题答案

这里给出每一章最后“测试题”和“动手试一试”中习题的答案。当然，有些问题的正确答案不止一个，特别是“动手试一试”中的习题，不过你可以通过这些答案来看你的思路是否正确。

第 1 章

测试题

1. 在 Windows 中，从“开始”菜单启动 IDLE，选择 Python 2.7 下面的 IDLE（Python GUI）。在 Mac OS X 上，如果你将 IDLE 放到了 Dock 上，则点击 Dock 中的 IDLE，或者双击 Applications 文件夹中 Python2.7 文件夹中的 IDLE.app。在 Linux 中，取决于你使用的窗口管理器，不过通常都会有一个 Applications 或 Programs 菜单。注意，在 Linux 中，很多人不用 IDLE。他们从终端中运行 Python，并使 vi 或 emacs 之类的编辑器编辑代码。
2. `print` 会在输出窗口中显示一些文本（在最前面的例子中，输出窗口就是 IDLE shell 窗口）。
3. Python 中的乘号是 `*`（星号）。
4. 运行程序时，IDLE 会显示这样一行：

```
>>> ======================== RESTART ========================
```

5. “执行”程序就是“运行”程序的另一种说法。

动手试一试

1. `>>> print 7 * 24 * 60`（一周有 7 天，一天有 24 小时，一小时有 60 分钟），所以答案应当是 10 080。

2. 你的程序应该类似这样：

```
print "My name is Warren Sande."
print "My birth date is January 1, 1970."
print "My favorite color is blue."
```

第 2 章

测试题

1. 可以在变量两边加上引号来告诉 Python 这个变量是一个字符串。
2. 这个问题就是：“可以改变赋给一个变量的值吗？”这要看你所说的“改变”是什么意思。如果有：

   ```
   myAge = 10
   ```

   就可以这样做：

   ```
   myAge = 11
   ```

   这样就改变了赋给 myAge 的内容。你把 myAge 标签移到了一个不同的东西上（从 10 移到了 11 上）。不过你并没有真正把 10 变成 11。所以更正确的说法应当是：你可以“把变量名重新指派到一个不同的值上”或者“为变量指定一个新的值”，而不是“改变变量的值”。
3. 不，TEACHER 与 TEACHEr 不同。因为变量名是区分大小写的，最后一个字母不同，所以这两个变量名也不同。
4. 对，'Blah' 和 "Blah" 是一样的。它们都是字符串，在这里，Python 并不关心使用的是单引号还是双引号，只要字符串左边的开始引号与右边的结束引号匹配就行。
5. 不，'4' 与 4 不同。第一个（'4'）是字符串（尽管这个字符串里只有一个字符），因为它两边加了引号。第二个（4）则是一个数。
6. 答案是 b。2Teacher 不是一个正确的变量名。Python 中的变量名不能以数字开头。
7. "10" 是一个字符串，因为它两边有引号。

动手试一试

1. 在交互模式中，可以这样做：

   ```
   >>> temperature = 25
   >>> print temperature
   25
   ```

2. 可以这样做：

   ```
   >>> temperature = 40
   >>> print temperature
   40
   ```

   或者这样做：

   ```
   >>> temperature = temperature + 15
   >>> print temperature
   40
   ```

3. 可以这样做：

```
>>> firstName = "Fred"
>>> print firstName
Fred
```

4. 如果使用变量，你的“每周有多少分钟”程序应该类似下面的代码：

```
>>> DaysPerWeek = 7
>>> HoursPerDay = 24
>>> MinutesPerHour = 60
>>> print DaysPerWeek * HoursPerDay * MinutesPerHour
10080
```

5. 要看如果一天有 26 小时会有什么结果，可以这样做：

```
>>> HoursPerDay = 26
>>> print DaysPerWeek * HoursPerDay * MinutesPerHour
10920
```

第 3 章

测试题

1. Python 使用 *（星号）表示乘法。
2. Python 会得出结果 8/3=2。因为 8 和 3 都是整数，所以 Python 2 会把答案向下取整为最接近的整数。（注意，在 Python 3 中，你会得出结果 2.66666666667，因为 Python 3 不像 Python 2 那样对整数默认做整除运算。）
3. 要得到余数，可以使用取余操作符：8 % 3。
4. 要得到 8/3 的小数结果，需要把其中一个数改为小数：8.0/3 或 8/3.0。（注意，在 Python 3 中，会自动得出小数结果。）
5. Python 中计算 6 * 6 * 6 * 6 的另一种做法是什么？ 6 ** 4
6. 17 000 000 采用 E 记法要写作 1.7e7。
7. 4.56e-5 就是 0.000 045 6。

动手试一试

解决这些问题还有其他方法。你可能会提出不同的方法来做这些事情。

1. (a) 计算每个人在餐厅要付多少钱：

```
>>> print 35.27 * 1.15 / 3
>>> 13.5201666667
```

把它四舍五入，每个人应当付 $13.52。

(b) 计算一个矩形的面积和周长：

```
length = 16.7
width = 12.5
Perimeter = 2 * length + 2 * width
Area = length * width
print 'Length = ', length, ' Width = ', width
print "Area = ", Area
print "Perimeter = ", Perimeter
```

下面是运行这个程序的示例输出：

```
Length = 16.7 Width = 12.5
Area = 208.75
Perimeter = 58.4
```

2. 下面是一个把华式度转换为摄氏度的程序：

```
fahrenheit = 75
celsius   = 5.0/9 * (fahrenheit - 32)
print "Fahrenheit  = ", fahrenheit, "Celsius =", celsius
```

3. 计算以某个速度行驶一定距离需要花多长时间：

```
distance = 200
speed = 80.0
time = distance / speed
print "time =", time
```

（要记住，除法中至少有一个数是小数，除非答案会向下取整为一个整数）。

第 4 章

测试题

1. `int()` 函数总是向下取整（这个数左边的最大整数）。

2. 在我们的温度转换程序中，可以这样做吗？

```
cel = float(5 / 9 * (fahr - 32))
cel = 5 / 9 * float(fahr - 32)
```

试试看，会发生什么：

```
>>> fahr = 75
>>> cel = float(5 / 9 * (fahr - 32))
>>> print cel
0.0
```

为什么不能正常工作？

要记住，括号里的一切会先完成。所以它会先这样：

```
75 - 32 = 43
```

然后再这样做：

```
5 / 9 = 0
```

因为它会从左到右计算，所以先完成 `5/9` 。因为 5 和 9 都是整数，所以 Python 会完成整除，将答案向下取整。由于这个答案小于 1，所以会取整为 0。然后得到：

```
0 * 43 = 0
```

接下来：

```
float(0) = 0.0
```

执行到 float() 时，已经太晚了——答案已经是 0 了！第二个公式也一样。

3. 可以"骗过" int()，让它四舍五入而不是向下取整，只需将传入 int() 的数加 0.5。

右面是一个例子（交互模式中）：

```
>>> a = 13.2
>>> roundoff = int(a + 0.5)
>>> roundoff
13
>>> b = 13.7
>>> roundoff = int(b + 0.5)
>>> b
14
```

如果原先的数小于 13.5，int() 会得到一个小于 14 的数，这会向下取整为 13。

如果原来的数大于或者等于 13.5，int() 会得到一个等于或者大于 14 的数，这就会向下取整为 14。

动手试一试

1. 可以使用 float() 将字符串转换为小数：

```
>>> a = float('12.34')
>>> print a
12.34
```

不过我们怎么能知道这是数而不是字符串呢？下面来检查类型：

```
>>> type(a)
<type 'float'>
```

2. 可以使用 int() 把小数转换为整数：

```
>>> print int(56.78)
56
```

结果会向下取整。

3. 可以使用 int() 把字符串转换为整数：

```
>>> a = int('75')
>>> print a
75
>>> type(a)
<type 'int'>
```

第 5 章

测试题

1. 对于这行代码：

```
answer = raw_input()
```

如果用户键入 12，answer 会包含一个字符串。这是因为 raw_input() 总是会得到一个字符串。

在一个小程序里试试看：

```
print "enter a number: ",
answer = raw_input()
print type(answer)

>>> ============== RESTART ==============
>>>
enter a number: 12
<type 'str'>
>>>
```

所以 raw_input() 会提供一个字符串。

记住，在 Python 3 中，raw、input() 改名为 input()3。

2. 要让 raw_input() 打印一条提示消息，可以在括号里的引号中加一些文本，如下：

```
answer = raw_input("Type in a number: ")
```

3. 要使用 raw_input() 得到一个整数，可以使用 int() 转换从 raw_input() 得到的字符串。这个工作可以分两步来完成，如下：

```
something = raw_input()
answer = int(something)
```

或者也可以一步完成，如下：

```
answer = int(raw_input())
```

4. 与上一题类似，只不过要使用 float() 而不是 int()。

动手试一试

1. 交互模式中，这个指令应当如下所示：

```
>>> first = 'Warren'
>>> last = 'Sande'
>>> print first + last
WarrenSande
```

唉呀！没有空格。可以在你的名字末尾加一个空格。

```
>>> first = 'Warren '
```

或者这样试试看：

```
>>> print first + ' ' + last
Warren Sande
```

还可以使用一个逗号，如下：

```
>>> first = 'Warren'
>>> last = 'Sande'
>>> print first, last
Warren Sande
```

2. 这个程序应当类似下面的代码：

```
first = raw_input('enter your first name: ')
last = raw_input('enter your last name: ')
print 'Hello,', first, last, 'how are you today?'
```

3. 这个程序应当类似下面的代码：

```
length = float(raw_input ('length of the room in feet: '))
width = float(raw_input ('width of the room in feet: '))
area = length * width
print 'The area is', area, 'square feet.'
```

4. 可以为上面第 3 题的程序增加几行代码：

```
length = float(raw_input ('length of the room in feet: '))
width = float(raw_input ('width of the room in feet: '))
cost_per_yard = float(raw_input ('cost per square yard: '))
area_feet = length * width
area_yards = area_feet / 9.0
total_cost = area_yards * cost_per_yard
print 'The area is', area_feet, 'square feet.'
print 'That is', area_yards, 'square yards.'
print 'Which will cost', total_cost
```

5. 程序应该类似下面的代码：

```
quarters = int(raw_input("How many quarters? "))
dimes = int(raw_input("How many dimes? "))
nickels = int(raw_input("How many nickels? "))
pennies = int(raw_input("How many pennies? "))
total = 0.25 * quarters + 0.10 * dimes + 0.05 * nickels + 0.01 * pennies
print "You nave a total of: ", total
```

第 6 章

测试题

1. 要用 EasyGui 显示一个消息框，可以使用 msgbox()，如下：

```
easygui.msgbox("This is the answer!")
```

2. 要用 EasyGui 得到一个字符串输入，要使用 enterbox。
3. 要得到整数输入，可以使用 enterbox（这会由用户得到一个字符串），然后把它转换为 int。或者也可以直接使用 integerbox。
4. 要从用户那里得到浮点数，可以使用一个 enterbox（这会提供一个字符串），然后使用 float() 函数把这个字符串转换成一个浮点数。
5. 默认值就像“自动获得的答案”。以下是一种可能使用默认值的情况：你在编写程序，你班里的所有学生都必须输入他们的名字和地址，你可以把你居住的城市名作为地址中的默认城市。这样一来，学生们就不用再键入城市了

(除非他们居住在其他城市)。

动手试一试

1. 以下是一个使用 EasyGui 的温度转换程序:

```
# tempgui1.py
# EasyGui version of temperature-conversion program
# converts Fahrenheit to Celsius
import easygui

easygui.msgbox('This program converts Fahrenheit to Celsius')
temperature = easygui.enterbox('Type in a temperature in Fahrenheit:')
Fahr = float(temperature)
Cel = (Fahr - 32) * 5.0 / 9
easygui.msgbox('That is ' + str(Cel) + ' degrees Celsius.')
```

2. 下面这个程序会询问你的名字以及地址的各个部分,然后显示完整的地址。要理解这个程序,如果了解如何强制换行会很有帮助:换行会让后面的文本从新的一行开始。为达到这个目的,需要使用 `\n`。这会在第 21 章解释,不过下面先提前了解一下:

```
# address.py
# Enter parts of your address and display the whole thing
import easygui
name = easygui.enterbox("What is your name?")
addr = easygui.enterbox("What is your street address?")
city = easygui.enterbox("What is your city?")
state = easygui.enterbox("What is your state or province?")
code = easygui.enterbox("What is your postal code or zip code?")

whole_addr = name + "\n" + addr + "\n" + city + ", " + state + "\n" + code

easygui.msgbox(whole_addr, "Here is your address:")
```

第 7 章

测试题

1. 输出将是:

```
Under 20
```

因为 my_number 小于 20,`if` 语句中的测试为 `true`,所以会执行 `if` 后面的块(这里只有一行代码)。

2. 输出将是:

```
20 or over
```

因为 my_number 大于 20,`if` 语句中的测试为 `false`,所以 `if` 后面的块代码不会执行。相反,会执行 `else` 块中的代码。

3. 要查看一个数是否大于 30 但小于或等于 40，可以使用下面的代码：

```
if number > 30  and  number <= 40:
    print 'The number is between 30 and 40'
```

你还可以这样做：

```
if 30 < number <= 40:
    print "The number is between 30 and 40"
```

4. 要检查字母“Q”是大写还是小写，可以这样做：

```
if answer == 'Q' or answer == 'q':
    print "you typed a 'Q' "
```

注意，我们打印的字符串使用了双引号，不过其中的“Q”两边是单引号。如果想知道如何打印引号，可以用另一种引号包围字符串。

动手试一试

1. 下面给出一个答案：

```
# program to calculate store discount
# 10% off for $10 or less, 20% off for more than $10
item_price = float(raw_input ('enter the price of the item: '))
if item_price <= 10.0:
    discount = item_price * 0.10
else:
    discount = item_price * 0.20
final_price = item_price - discount
print 'You got ', discount, 'off, so your final price was', final_price
```

这里没有考虑把答案四舍五入为两位小数（美分），也没有显示美元符。

2. 以下给出一种做法：

```
# program to check age and gender of soccer players
# accept girls who are 10 to 12 years old
gender = raw_input("Are you male or female? ('m' or 'f') ")
if gender  == 'f':
    age = int(raw_input('What is your age? '))
    if age >= 10 and age <= 12:
        print 'You can play on the team'
    else:
        print 'You are not the right age.'
else:
    print 'Only girls are allowed on this team.'
```

3. 以下给出一个答案：

```
# program to check if you need gas.
# Next station is 200 km away
tank_size = int(raw_input('How big is your tank (liters)? '))
full = int(raw_input ('How full is your tank (eg. 50 for half full)?'))
mileage = int(raw_input ('What is your gas mileage (km per liter)? '))
range = tank_size * (full / 100.0) * mileage
print 'You can go another', range, 'km.'
print 'The next gas station is 200km away.'
if range <= 200:
    print 'GET GAS NOW!'
else:
    print 'You can wait for the next station.'
```

要增加一个 5 公升的缓冲区，需要把这行代码：

```
range = tank_size * (full / 100.0) * mileage
```

改为：

```
range = (tank_size - 5) * (full / 100.0) * mileage
```

4. 下面是一个简单的口令程序：

```
password  = "bigsecret"
guess  = raw_input("Enter your password: ")
if guess == password:
    print "Password correct.  Welcome"
    # put the rest of the code for your program here
else:
    print "Password incorrect.  Goodbye"
```

第 8 章

测试题

1. 这个循环会运行 5 次。
2. 这个循环会运行 3 次，i 的值分别是 i = 1, i = 3, i = 5。
3. range(1, 8) 会给出 [1, 2, 3, 4, 5, 6, 7]。
4. range(8) 会给出 [0, 1, 2, 3, 4, 5, 6, 7]。
5. range(2, 9, 2) 会给出 [2, 4, 6, 8]。
6. range (10, 0, -2) 会给出 [10, 8, 6, 4, 2]。
7. 可以使用 continue 停止一个循环的当前迭代，直接跳到下一次迭代。
8. while 循环会在测试的条件为 false 时停止。

动手试一试

1. 下面的程序使用一个 for 循环打印用户选择的乘法表：

```
# program to print multiplication table up to 10
number = int(raw_input('Which table would you like? '))
print 'Here is your table:'
for i in range(1, 11):
    print number, 'x', i, '=', number * i
```

2. 下面的程序使用 while 循环打印同一个乘法表：

```
# program to print mult table (while loop)
number = int(raw_input('Which table would you like? '))
print 'Here is your table:'
i = 1
while i <= 10:
    print number, 'times', i, '=', number * i
    i = i + 1
```

3. 下面的程序会根据用户定义的范围打印乘法表：

```
# program to print multiplication table
# user inputs how high they want it to go
number = int(raw_input('Which table would you like? '))
limit = int(raw_input('How high would you like it to go? '))
print 'Here is your table:'
for i in range(1, limit + 1):
    print number, 'times', i, '=', number * i
```

注意 for 代码行中 range() 的第二项包含一个变量，而不是一个数。我们将在第 11 章介绍有关的更多内容。

第 9 章

动手试一试

1. 下面是我为温度转换程序增加的一些注释：

```
# tempconv1.py
# program to convert a Fahrenheit temperature to Celsius
Fahr = 75
Cel = (Fahr - 32) * 5.0 / 9   #decimal division, not integer
print "Fahrenheit = ", Fahr, "Celsius = ", Cel
```

第 10 章

动手试一试

你有没有试着键入这个程序并运行它？不要忘记把图片放在程序所在的同一个文件夹中。

第 11 章

测试题

1. Python 中可以在 `range()` 函数中放一个变量来建立可变循环，

   如下：
   ```
   for i in range(numberOfLoops)
   ```
   或者：
   ```
   for i in range(1, someNumber)
   ```
2. 要建立嵌套循环，需要把一个循环放在另一个循环的循环体中，如下：

   ```
   for i in range(5):
       for j in range(8):
           print "hi",
       print
   ```

   这个代码会打印 5 行（外循环），每一行上打印 8 次 `"hi"`（内循环）。
3. 将会打印 15 个星号。
4. 这个代码的输出如下所示：
   ```
   * * *
   * * *
   * * *
   * * *
   * * *
   ```
5. 对于 4 层的判定树，会有 2**4 或者 2 * 2 * 2 * 2 种可能的选择。也就是 16 种可能的选择，或者决策树有 16 条路径。

动手试一试

1. 下面给出这个倒计时定时器程序，它会询问用户从哪里开始：

   ```
   # Countdown timer asks the user where to start
   import time
   start = int(raw_input("Countdown timer:  How many seconds? ", ))
   for i in range (start, 0, -1):
       print i
       time.sleep(1)
   print "BLAST OFF!"
   ```

2. 下面这个程序会在各个数旁边打印一行星号：

```
# Countdown timer asks the user where to start
# and prints stars beside each number

import time
start = int(raw_input("Countdown timer:  How many seconds? ", ))
for i in range (start, 0, -1):
    print i,
    for star in range(i):
        print '*',
    print
    time.sleep(1)
print "BLAST OFF!"
```

第 12 章

测试题

1. 可以使用 append()、insert() 或 extend() 向列表增加元素。
2. 可以使用 remove()、pop() 或 del() 从列表删除元素。
3. 要得到列表的一个有序副本，可以采用下面任意一种做法：
   - 建立列表的副本，使用分片：new_list = my_list[:]，然后对新列表排序：new_list.sort()
   - 使用 sorted() 函数：new_list = sorted(my_list)
4. 使用 in 关键字可以得出一个特定值是否在一个列表中。
5. 使用 index() 方法可以得出一个值在列表中的位置。
6. 元组是一个与列表类似的集合，只不过元组不能改变。元组是不可改变的，而列表是可改变的。
7. 可以采用多种方法建立一个双重列表。
   - 使用嵌套的中括号：

```
>>> my_list =[[1,2,3],['a','b','c'],['red','green',blue']]
```

   - 使用 append()，并追加一个列表：

```
>>> my_list = []
>>> my_list.append([1, 2, 3])
>>> my_list.append(['a', 'b', 'c'])
>>> my_list.append(['red', 'green', 'blue'])
>>> print my_list
[[1, 2, 3], ['a', 'b', 'c'], ['red', 'green', 'blue']]
```

❒ 建立单个列表，再合并这些列表：

```
>>> list1 = [1, 2, 3]
>>> list2 = ['a', 'b', 'c']
>>> list3 = ['red', 'green', 'blue']
>>> my_list = [list1, list2, list3]
>>> print my_list
[[1, 2, 3], ['a', 'b', 'c'], ['red', 'green', 'blue']]
```

8. 可以使用两个索引得到双重列表中的一个值：

```
my_list = [[1, 2, 3], ['a', 'b', 'c'], ['red', 'green', 'blue']]
my_color = my_list[2][1]
```

这个答案是 'green'。

9. 字典是键值对的集合。

10. 你可以通过指定键和值的方式在字典中添加条目：

```
phone_numbers['John'] = '555-1234'
```

11. 要通过键在字典中查找一个条目，可以使用索引：

```
print phone_numbers['John']
```

动手试一试

1. 下面这个程序会得到 5 个名字，把它们放在一个列表中，然后打印出来：

```
nameList = []
print "Enter 5 names (press the Enter key after each name):"
for i in range(5):
    name = raw_input()
    nameList.append(name)

print "The names are:", nameList
```

2. 下面这个程序会打印原来的列表和排序后的列表：

```
nameList = []
print "Enter 5 names (press the Enter key after each name):"
for i in range(5):
    name = raw_input()
    nameList.append(name)

print "The names are:", nameList
print "The sorted names are:", sorted(nameList)
```

3. 下面这个程序只打印列表中的第 3 个名字：

```
nameList = []
print "Enter 5 names (press the Enter key after each name):"
for i in range(5):
    name = raw_input()
    nameList.append(name)

print "The third name is:", nameList[2]
```

4. 下面这个程序允许用户替换列表中的一个名字：

```
nameList = []
print "Enter 5 names (press the Enter key after each name):"
for i in range(5):
    name = raw_input()
    nameList.append(name)
print "The names are:", nameList
print "Replace one name.  Which one? (1-5):",
replace = int(raw_input())
new = raw_input("New name: ")
nameList[replace - 1] = new
print "The names are:", nameList
```

5. 下面这个程序允许用户创建包含单词和对应定义的词典：

```
user_dictionary = {}
while 1:
    command=raw_input("'a'to add word,'l'to lookup a word,'q'to quit")

    if command == "a":
        word = raw_input("Type the word: ")
        definition = raw_input("Type the definition: ")
        user_dictionary[word] = definition
        print "Word added!"

    elif command == "l":
        word = raw_input("Type the word: ")
        if word in user_dictionary.keys():
            print user_dictionary[word]
        else:
            print "That word isn't in the dictionary yet."

    elif command == 'q':
        break
```

第 13 章

测试题

1. 使用 def 关键字来创建一个函数。
2. 调用函数时要使用函数名和一对小括号。
3. 调用函数时把参数放在小括号里，就可以向这个函数传入参数。
4. 一个函数可以有任意多个参数，对此没有任何限制。
5. 函数使用 return 关键字向调用者发回信息。
6. 函数完成运行后，所有局部变量都会撤销。

动手试一试

1. 这个函数只需要一组 print 语句：

```
def printMyNameBig():
    print "  CCCC       A       RRRRR  TTTTTTT  EEEEEE  RRRRR  "
    print " C    C     A A      R    R    T     E       R    R "
    print "C          A   A     R    R    T     EEEE    R    R "
    print "C         AAAAAAA    RRRRR     T     E       RRRRR  "
    print " C    C  A       A   R    R    T     E       R    R "
    print "  CCCC  A         A  R     R   T     EEEEEE  R     R"
```

调用这个函数的程序如下所示：

```
for i in range(5):
    printMyNameBig()
```

2. 下面给出我的做法，这里利用 7 个参数打印地址：

```
# define a function with seven arguments
def printAddr(name, num, street, city, prov, pcode, country):
    print name
    print num,
    print street
    print city,
    if prov !="":
        print ", "+prov
    else:
        print ""
    print pcode
    print country
    print

#call the function and pass seven arguments to it
printAddr("Sam","45", "Main St.","Ottawa", "ON", "K2M 2E9","Canada")
printAddr("Jian", "64", "2nd Ave.","Hong Kong", "", "235643","China")
```

3. 没有具体答案，可以动手试一试。

4. 合计零钱的函数应当如下所示：

```
def addUpChange(quarters, dimes, nickels, pennies):
    total  = 0.25 * quarters + 0.10 * dimes + 0.05 * nickels + 0.01 *
        pennies
    return total
```

调用它的程序可能如下所示：

```
quarters = int(raw_input("quarters: "))
dimes = int(raw_input("dimes: "))
nickels = int(raw_input("nickels: "))
pennies = int(raw_input("pennies: "))

total  = addUpChange(quarters, dimes, nickels, pennies)

print "You have a total of: ", total
```

第 14 章

测试题

1. 要定义一个新的对象类型，需要使用 class 关键字。
2. 属性是有关一个对象“你所知道的信息”，就是包含在对象中的变量。
3. 方法是可以对对象做的“动作”，就是包含在对象中的函数。
4. 类只是对象的定义或蓝图，从这个蓝图建立对象时得到的就是实例。
5. 在对象方法中通常用 self 作为实例引用。
6. 多态是指不同对象可以有同名的两个或多个方法。这些方法可以根据它们所属的对象有不同的行为。
7. 继承是指对象能够从它们的“双亲”（父类）得到属性和方法。“子”类（也称为子类或派生类）会得到父类的所有属性和方法，还可以有父类所没有的属性和方法。

动手试一试

1. 对应银行账户的类如下所示：

```
class BankAccount:
    def __init__(self, acct_number, acct_name):
        self.acct_number = acct_number
        self.acct_name = acct_name
        self.balance = 0.0
```

```
    def displayBalance(self):
        print "The account balance is:", self.balance

    def deposit(self, amount):
        self.balance = self.balance + amount
        print "You deposited", amount
        print "The new balance is:", self.balance

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance = self.balance - amount
            print "You withdrew", amount
            print "The new balance is:", self.balance
        else:
            print "You tried to withdraw", amount
            print "The account balance is:", self.balance
            print "Withdrawal denied.  Not enough funds."
```

下面的代码用来测试这个类，确保它能正常工作：

```
myAccount = BankAccount(234567, "Warren Sande")
print "Account name:", myAccount.acct_name
print "Account number:", myAccount.acct_number
myAccount.displayBalance()

myAccount.deposit(34.52)
myAccount.withdraw(12.25)
myAccount.withdraw(30.18)
```

2. 要建立一个利息账户，需要建立一个 BankAccount 子类，并创建一个方法来增加利息：

```
class InterestAccount(BankAccount):
    def __init__(self, acct_number, acct_name, rate):
        BankAccount.__init__(self, acct_number, acct_name)
        self.rate = rate

    def addInterest (self):
        interest = self.balance * self.rate
        print"adding interest to the account,",self.rate*100,"percent"
        self.deposit (interest)
```

下面是一些测试代码：

```
myAccount = InterestAccount(234567, "Warren Sande", 0.11)
print "Account name:", myAccount.acct_name
print "Account number:", myAccount.acct_number
myAccount.displayBalance()
myAccount.deposit(34.52)
myAccount.addInterest()
```

第 15 章

测试题

1. 使用模块有下面这些好处：
   - 可以只写一次代码，并在多个程序中使用；
   - 你可以使用其他人写的模块；
   - 代码文件会更小，所以更容易发现代码中的问题；
   - 可以只使用完成工作真正需要的部分（模块）。
2. 要创建模块，需要编写一些 Python 代码并保存在文件中。
3. 想使用一个模块时，要用 import 关键字导入。
4. 导入模块与导入命名空间是一样的。
5. 导入 time 模块从而能访问这个模块中的所有名字有两种方法，

   分别是：
   ```
   import time
   ```
   和
   ```
   from time import *
   ```

动手试一试

1. 要编写一个模块，只需要把“用大写字母打印名字”函数中的代码放在一个文件中，比如 bigname.py 文件。然后，要导入这个模块并调用函数，

   可以这样做：
   ```
   import bigname
   bigname.printMyNameBig()
   ```
   也可以这样做：
   ```
   from bigname import *
   printMyNameBig()
   ```
2. 要把 c_to_f() 导入主程序的命名空间，可以这样做：
   ```
   from my_module import c_to_f
   ```
   或者这样做：
   ```
   from my_module import *
   ```
3. 下面这个小程序会打印从 1 到 20 的 5 个随机整数：
   ```
   import random
   for i in range(5):
       print random.randint(1, 20)
   ```
4. 下面这个小程序会工作 30 秒，每 3 秒打印一个随机小数：

```
import random, time
for i in range(10):
    print random.random()
    time.sleep(3)
```

第 16 章

测试题

1. RGB 值 [255, 255, 255] 得到白色。
2. RGB 值 [0, 255, 0] 得到绿色；
3. 要画矩形，可以使用 Pygame 方法 `pygame.draw.rect()`；
4. 要画线把多个点连在一起（如连连看），可以使用 `pygame.draw.lines()` 方法。
5. “像素”是“图像元素”的简写，表示屏幕上（或纸上）的一个点。
6. 在一个 Pygame 窗口中，位置 [0, 0] 位于左上角。
7. 在这个图中，位置 [50, 200] 位于字母 B。
8. 在这个图中，位置 [300, 50] 位于字母 D。
9. 可以使用 `blit()` 方法在 Pygame 中复制图像。
10. 要移动一个图像或者完成动画，可以使用以下两个步骤：
    - 从原来的位置擦除图像；
    - 在新位置上绘制图像。

动手试一试

1. 下面的程序会在屏幕上画出一些不同的形状。也可以在 \answers 文件夹和网站上找到 TIO_CH16_1.py。

```
import pygame, sys
pygame.init()
screen=pygame.display.set_mode((640, 480))
screen.fill((250, 120, 0))
pygame.draw.arc(screen,(255,255,0),pygame.rect.Rect(43,368,277,235),
   -6.25, 0, 15)
pygame.draw.rect(screen,(255, 0, 0),pygame.rect.Rect(334,191,190,290))
pygame.draw.rect(screen,(128, 64, 0),pygame.rect.Rect(391,349,76,132))
pygame.draw.line(screen,(0, 255, 0),(268, 259),(438,84),25)
pygame.draw.line(screen,(0, 255, 0),(578, 259),(438,84),25)
pygame.draw.circle(screen, (0, 0, 0), (452, 409), 11, 2)
pygame.draw.polygon(screen, (0, 0, 255),[(39, 39), (44, 136),(59,136),
   (60,102),(92,102),(94,131),(107, 141), (111, 50), (97, 50), (93,
   86), (60, 82), (58, 38)], 5)
pygame.draw.rect(screen,(0,0,255), pygame.rect.Rect(143,90,23,63),5)
pygame.draw.circle(screen, (0, 0, 255), (153, 60), 15, 5)
clock = pygame.time.Clock()
pygame.display.flip()
```

```
running = True
while running:
    clock.tick(60)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type==pygame.KEYDOWN and event.key==pygame.K_ESCAPE:
            running = False
pygame.quit()
```

2. 要把沙滩球图像换成一个不同的图像，只需把这行代码中的文件名：

```
my_ball = pygame.image.load('beach_ball.png')
```

替换成另一个图片的文件名。

3. 在代码清单 16-16 中，只需把

```
x_speed = 10
y_speed = 10
```

改为其他的值，如

```
x_speed = 20
y_speed = 8
```

4. 要让球在一面“隐形”的墙上反弹，可以把代码清单 16-16 中的这行代码

```
if x > screen.get_width() - 90  or  x < 0:
```

改为：

```
if x > screen.get_width() - 250  or  x < 0:
```

这会让球在到达窗口边界之前就反向。可以对 y 坐标做同样的处理，让它在到达“地板”时也会反弹。

5. 将代码清单 16-6 中的 display.flip 移到 `while` 循环内部，并增加一个延迟之后，代码如下所示：

```
import pygame, sys, random
pygame.init()
screen = pygame.display.set_mode([640,480])
screen.fill([255, 255, 255])
for i in range (100):
    width = random.randint(0, 250)
    height = random.randint(0, 100)
    top = random.randint(0, 400)
    left = random.randint(0, 500)
    pygame.draw.rect(screen, [0,0,0], [left, top, width, height], 1)
    pygame.display.flip()
    pygame.time.delay(30)
```

应该能看到各个矩形会单独出现，因为我们放慢了程序的速度，现在画出各

个矩形之后会刷新显示。如果对正弦曲线程序做这个处理，可以看到正弦曲线中的各个点分别画出。

第 17 章

测试题

1. 矩形碰撞检测是指使用对象的一个外围矩形来检测两个图形对象是否接触或重叠。
2. 像素完美碰撞检测是指使用图形对象的实际轮廓来完成碰撞检测。矩形碰撞检测使用对象的外围矩形来确定碰撞。像素完美碰撞检测更准确更真实，不过也需要写更多代码，另外还会让程序速度减慢。
3. 可以使用常规的 Python 列表或 Pygame 动画精灵组来跟踪多个精灵对象。
4. 代码中可以在各帧之间增加延迟来控制动画的速度（帧速率），或者使用 `pygame.time.Clock` 得到某个帧速率。还可以改变每一帧将对象移动多远（多少像素）。
5. 使用 `delay` 方法不太准确，因为它没有考虑每一帧代码本身所花费的时间，所以你不能准确地知道最终的帧速率。
6. 可以使用 `pygame.time.Clock.get_fps()` 得到程序运行的帧速率。

第 18 章

测试题

1. 程序可以响应的两种事件分别是键盘事件和鼠标事件。
2. 处理事件的代码称为事件处理器。
3. Pygame 使用 `KEYDOWN` 事件来检测按键是否按下。
4. `pos` 属性会指出事件发生时鼠标所在的位置。
5. 要为用户事件得到下一个可用的事件编号，可以使用 `pygame.NUMEVENTS`。
6. 要创建一个定时器，可以使用 `pygame.time.set_timer()`。
7. 要在 Pygame 窗口中显示文本，可以使用 `font` 对象。
8. 使用字体对象有 3 个步骤：
   - ❒ 创建一个字体对象；
   - ❒ 渲染文本，创建一个表面；
   - ❒ 把这个表面块移到显示表面。

动手试一试

1. 如果球没有碰到球拍的顶边，而是碰到了球拍的左右两边，为什么会有奇怪

的表现？这是因为这里有一个碰撞，所以代码尝试让球的 y 方向反向（让它向上而不是向下）。但是因为球是从两边（左边或右边）过来的，即使在反向之后它仍会与球拍“碰撞”。下一次循环（一帧之后）时，它会再次反向，因此会再次向下，如此继续。要解决这个问题，有一种简单的方法：当球与球拍碰撞时总是将球设置为向“上”（y 速度是一个负值）。这不能算是一种完美的解决办法，因为这意味着即使球碰到球拍左右两边也会向上反弹——这可不太真实！不过这样能解决球在球拍两边来回反弹的问题。如果你想要一种更真实的解决方案，可能需要多写一些代码。也许要增加一些内容，在“反弹”之前检查球碰到了球拍的哪一边。

2. 我们已经在网站上给出了有关代码的一个例子，可以为程序增加随机性，见 TIO_CH18_2.py。

第 19 章

测试题

1. 存储声音的文件类型包括波形文件（.wav）、MP3（.mp3）、Ogg Vorbis 文件（.ogg）和 Windows 媒体音频文件（.wma）。
2. `pygame.mixer` 模块用来播放音乐。
3. 要用各个声音对象的 `set_volume()` 方法设置 Pygame 声音对象的音量。
4. 使用 `pygame.mixer.music.set_volume()` 设置背景音乐的音量。
5. 要让音乐淡出，可以使用 `pygame.mixer.music.fadeout()` 方法。要提供淡出时间（毫秒数，即千分之一秒）作为参数。例如 `pygame.mixer.music.fadeout(2000)` 会让声音在 2 秒内淡出。

动手试一试

我们已经在网站上提供了加入声音的猜数程序的代码，见 TIO_CH19_1.py。

第 20 章

测试题

1. GUI 图形元素有 3 个名字，分别是控件、部件和组件。
2. 要进入一个菜单，与 Alt 同时按下的字母叫做热键。
3. Qt Designer 文件要以 .ui 结尾。
4. 使用 PyQt 的 GUI 中可以包含以下组件类型：按钮、复选框、进度条、列表、单选钮组、微调框、滑动条、文本域、图像、标签以及很多其他组件。查看 Qt Designer 的 Widget Box，可以看到全部组件类型。

5. 要让组件完成某项工作，需要一个事件处理器。
6. 在 Qt Designer 中要使用 &（与字符）定义热键。
7. Qt Designer 中微调框的内容总是一个整数。

动手试一试

1. 我们已经在网站上给出了使用 Qt Designer 完成的猜数程序，见 TIO_CH20_1.py 和 TIO_CH20_1.rsrc.ui。
2. 要解决这个微调框问题，需要在 Qt Designer 中选择微调组件。在属性编辑器中改变 minimum 和 maximum 属性。minimum 属性应当取一个很小的值，比如 –10x00，maximum 可以非常大，比如 1000000。

第 21 章

测试题

1. 如果有两个单独的 print 语句，而且希望所有内容都打印在同一行上，可以在第一个 print 语句的末尾加一个逗号，如下：

```
print "What is",
print "your name?"
```

2. 打印时要增加额外的空行，可以另外增加 print 语句（其中不含任何内容），如下：

```
print "Hello"
print
print
print
print "World"
```

也可以打印换行符 \n，如下：

```
print "Hello\n\n\nWorld"
```

3. 要让内容按列对齐，可以使用制表符 \t。
4. 要用 E 记法打印一个数，需要使用格式字符串 %e 或 %E，如下：

```
>>> number = 12.3456
>>> print '%e' % number
1.234560e+001
```

动手试一试

1. 这种程序应该像这样：

```
name = raw_input("What is your name? ")
age = int(raw_input("How old are you? "))
color = raw_input("What is your favorite color? ")

print "Your name is", name,
print "you are ", age, "years old,",
print "and you like the color", color
```

2. 使用制表符让乘法表对齐的代码如下：

```
for looper in range(1, 11):
    print looper, "\ttimes 8 =\t", looper * 8
```

注意单词 times 前面和 = 号后面的 \t。

3. 下面的程序会打印 8 的各个分数：

```
for i in range(1, 9):
    fraction = i / 8.0
    print str(i) + '/8 = %.3f' % fraction
```

第一部分 print str(i) + '/8 = 打印分数。最后一部分 %.3f' % fraction，打印小数结果（带 3 个小数位）。

第 22 章

测试题

1. Python 中用来处理文件的对象称为文件对象。
2. 要使用 open() 函数创建文件对象，这是 Python 的内置函数之一。
3. 文件名是磁盘上（或其他存储介质，如 flash 盘）存储文件时使用的名字。Python 中处理文件时要使用文件对象。文件对象名与磁盘上的文件名不必相同。
4. 程序完成文件的读写后，应当关闭文件。
5. 如果以追加模式打开一个文件，并在文件中写入内容，你写入的信息会增加（追加）到文件末尾。
6. 如果以写模式打开一个文件，然后在文件中写入内容，文件中原来的所有内容都会丢失，替换为新的数据。
7. 要重置为从文件起始位置开始读，可以使用 seek() 方法，并传入参数 0，如下：

```
myFile.seek(0)
```

8. 使用 pickle 把 Python 对象保存到文件时，可以使用 pickle.dump() 方法，

并指定希望保存的对象以及文件名作为参数，如下：

```
pickle.dump(myObject, "my_pickle_file.pkl")
```

9. 要从 pickle 文件还原或获取对象，可以使用 `pickle.load()` 方法，指定 pickle 文件作为参数，如下：

```
myObject = pickle.load("my_pickle_file.pkl")
```

动手试一试

1. 下面是创建滑稽句子的一个简单程序：

```
import random

noun_file = open("nouns.txt", 'r')
nouns = noun_file.readline()
noun_list = nouns.split(',')
noun_file.close()

adj_file = open("adjectives.txt", 'r')
adjectives = adj_file.readline()
adj_list = adjectives.split(',')
adj_file.close()

verb_file = open("verbs.txt", 'r')
verbs = verb_file.readline()
verb_list = verbs.split(',')
verb_file.close()

adverb_file = open("adverbs.txt", 'r')
adverbs = adverb_file.readline()
adverb_list = adverbs.split(',')
adverb_file.close()

noun = random.choice(noun_list)
adj = random.choice(adj_list)
verb = random.choice(verb_list)
adverb = random.choice(adverb_list)

print"The", adj, noun, verb, adverb + '.'
```

单词文件应当是用逗号分隔的单词列表。

2. 下面的程序会把一些数据保存在文本文件中：

```
name = raw_input("Enter your name: ")
age = raw_input("Enter your age: ")
color = raw_input("Enter your favorite color: ")
food = raw_input("Enter your favorite food: ")
```

```
my_data = open("my_data_file.txt", 'w')
my_data.write(name + "\n")
my_data.write(age + "\n")
my_data.write(color + "\n")
my_data.write(food)

my_data.close()
```

3. 下面的程序使用 pickle 模块保存一些数据：

```
import pickle

name = raw_input("Enter your name: ")
age = raw_input("Enter your age: ")
color = raw_input("Enter your favorite color: ")
food = raw_input("Enter your favorite food: ")

my_list = [name, age, color, food]

pickle_file = open("my_pickle_file.pkl", 'w')
pickle.dump(my_list, pickle_file)

pickle_file.close()
```

第 23 章

测试题

1. 随机事件是指可能发生的一些事情（“事件”），你事先不知道会有什么结果。扔硬币和掷骰子就是随机事件的两个例子。扔硬币时你不知道它会面朝上还是面朝下，扔一对骰子时你不知道最后得到的总数是什么。
2. 扔一个 11 面的骰子与扔两个 6 面的骰子不同，因为对于一个 11 面的骰子，所有数（从 2 到 12）出现的概率是一样的。而对于两个 6 面的骰子，有些数（两个骰子的总数）出现的概率会大于另外一些数。
3. 在 Python 中模拟扔骰子有两种方法：

```
import random
sides = [1, 2, 3, 4, 5, 6]
die_1 = random.choice(sides)
```

和

```
import random
die_1 = random.randint(1, 6)
```

4. 我们使用了一个对象来表示一张牌。

5. 我们使用了一个列表来表示一副牌，列表中的每一项都是一张牌（一个对象）。
6. 要从一副牌或一手牌中删除一张牌，我们使用了列表的 `remove()` 方法，如 `deck.remove()` 或 `hand.remove()`。

动手试一试

直接动手试一试，看看会发生什么。

第 24 章

测试题

1. 使用计算机仿真有这样一些原因：
   - 省钱（真实世界里有些实验的成本太高，这些实验就可以利用计算机仿真来完成）；
   - 保护人和设备（真实世界里有些实验可能很危险，这些实验就可以借助计算机仿真来完成）；
   - 尝试一些在真实世界中不可能的事情（比如说让一个比较大的小行星撞击月球）；
   - 让时间加快（使实验比真实世界中的实际实验更快），这对于研究一些可能花很长时间才能完成的事情（比如冰河融化）很有帮助；
   - 让时间放慢（使实验比真实世界中的实际实验更慢），这对于研究一些可能发生太快的事情很有帮助，比如电子信号在线路中的传送。.
2. 你可以列出你想到的任何类型的计算机仿真。可以是游戏、数学或科学程序，甚至也可以是天气预报（这也是用计算机仿真创建的）。
3. 要使用 `timedelta` 对象存储两个日期或时间之差。

动手试一试

这一节的程序都非常长——确实太长了，所以不再在本书中列出。你可以在网站上找到所有相关的文件：

TIO_CH24_1.py——提供脱离轨道检查的 Lunar Lander；

TIO_CH24_2.py——增加重玩选项的 Lunar Lander；

TIO_CH24_3.py——增加 pause 按钮的电子宠物 GUI。

第 26 章

动手试一试

1. 这是一个可以打败 CivcleAI 的机器人：

```
class AI:
    def __init__(self):
        self.isFirstTurn = True
    def turn(self):
        if self.isFirstTurn:
            self.robot.turnLeft()
            self.isFirstTurn = False
        elif self.robot.lookInFront() == "bot":
            self.robot.attack()
        else:
            self.robot.doNothing()
```

这个机器人的策略是等待 CircleAI 绕圈，等它在前方时攻击它。我能写出这个机器人，是因为我知道 CircleAI 的工作方式，它并不能打败其他的机器人。如我在本章正文中所说，要创建一个必胜的机器人是非常非常困难的，尤其是在连对手是什么机器人都不知道的情况下。

版 权 声 明

父与子的编程之旅

与小卡特一起学Python

编程是一项充满乐趣的挑战，想要上手也非常容易！本书中，Warren和Carter父子以亲切的笔调、通俗的语言，透彻全面地介绍了计算机编程世界。他们以简单易学的Python语言为例，通过可爱的漫画、有趣的例子，生动地介绍了变量、循环、输入和输出、数据结构以及图形用户界面等编程的基本概念。只要懂得计算机的基本操作，如启动程序、保存文件，任何人都可以跟随本书，由简入难，学会编写程序，甚至制作游戏。

本书内容经过教育专家的评审，经过孩子的亲身检验，并得到了家长的认可。如果你想教孩子学习计算机编程，培养孩子思考问题和解决问题的能力，那就赶快拿起本书，展开亲子学习之旅吧！如果你想自己学习编程，也赶紧拿起本书，探索看似神秘的编程世界吧！



“通过这本书来学习编程，真是再简单不过了！”

——Shawn Stebner，
英特尔公司网络工程师

“上到88岁，下到8岁，任何想学习编程的人都可以阅读这本书。它不仅以一种有趣的方式介绍了Python编程，而且其中的最佳实践还适用于其他编程语言的学习。”

——Ben Ooms，Sogeti公司软件工程师

“Python是我向编程新手推荐的首选语言，而这本书正是非常好的学习资源。第1版出版后，我就一直向学生们推荐它。”

——Dave Briccetti，软件开发者，教师

“如果你想学习编程，或者想教孩子编程，那么这本书就是你的不二选择。”

——Cuberick.com

插图 Martin Murtonen

MANNING

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机/程序设计

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-36717-4

ISBN 978-7-115-36717-4
定价：69.00元

看完了

如果您对本书内容有疑问，可发邮件至contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这里可以找到我们：

微博 @图灵教育 : 好书、活动每日播报
微博 @图灵社区 : 电子书和好文章的消息
微博 @图灵新知 : 图灵教育的科普小组
微信 图灵访谈 : ituring_interview，讲述码农精彩人生
微信 图灵教育 : turingbooks